2

Finite Automata

Formal Definitions

A *finite automaton* is a structure

$$M = (Q, \Sigma, \Delta, s, F)$$

where

- Q is a finite set of states

- Σ is a finite alphabet

- $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation: $\langle p, a, q \rangle \in \Delta$ when there is a transition from state p to state q labeled by the symbol a

- $s \in Q$ is the start state

- $F \subseteq Q$ is a set of final states.

Finite automaton M *accepts* string $u = a_1 \ldots a_k$ if there is a path in M starting with s labeled by $a_1, a_2, \ldots, a_k$, and ending up in a final state.

Formally: $M = (Q, \Sigma, \Delta, s, F)$ accepts $u = a_1 \ldots a_k$ if there exists $q_0, q_1, \ldots, q_k \in Q$ such that $q_0 = s$, $q_k \in F$, and $\langle q_{i-1}, a_i, q_i \rangle \in \Delta$ for all $1 \leq i \leq k$.

The language accepted by M is

$$L(M) = \{u \mid M \text{ accepts } u\}$$

Example: The following finite automaton accepts exactly the strings over $\{a, b\}$ of even length:
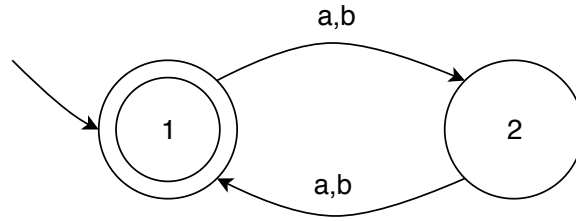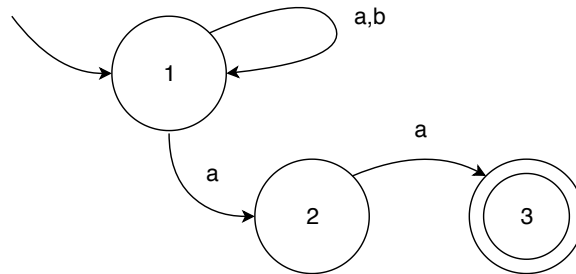
$$M_{even} = (\{1, 2\}, \{a, b\}, \Delta_{even}, 1, \{1\})$$

where the transitions go from state 1 to state 2 and back no matter the symbol:

$$\Delta_{even} = \{\langle 1, a, 2 \rangle, \langle 1, b, 2 \rangle, \langle 2, a, 1 \rangle, \langle 2, b, 1 \rangle\}$$

Example: The following finite automaton accepts exactly the strings over $\{a, b\}$ whose last two symbols are a:

$$M_{last} = (\{1, 2, 3\}, \{a, b\}, \Delta_{last}, 1, \{3\})$$

where

$$\Delta_{last} = \{\langle 1, a, 1 \rangle, \langle 1, b, 1 \rangle, \langle 1, a, 2 \rangle, \langle 2, a, 3 \rangle\}$$

Theorem: A language A is accepted by some finite automaton M if and only if A is regular.

One direction of the equivalence is pretty easy, namely showing that when A is regular there is some finite automaton that accepts A.

Since A is regular, this means that there is a regular expression r with $[\![r]\!] = A$. We can define a recursive procedure that constructs a finite automaton from a regular expression and that accepts the language of the regular expression. The procedure will recurse over the structure of the regular expression.

For the bases case of the recursion, it is easy to construct finite automata for $[\![0]\!] = \varnothing$:

$$M_0 = (\{1\}, \Sigma, \varnothing, 1, \varnothing)$$

for $[\![1]\!] = \{\epsilon\}$:

$$M_1 = (\{1\}, \Sigma, \varnothing, 1, \{1\})$$

and for $[\![\mathsf{a}]\!] = \{\mathsf{a}\}$:

$$M_{\mathsf{a}} = (\{1, 2\}, \Sigma, \{\langle 1, \mathsf{a}, 2\rangle\}, 1, \{2\})$$

For the recursive cases, first consider $r_1 + r_2$. If r_1 and r_2 are regular expressions, by recursion we can obtain finite automata $M_1 = (Q_1, \Sigma, \Delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \Delta_2, s_2, F_2)$ that accept the languages of r_1 and r_2 respectively. Without loss of generality, we can take Q_1 and Q_2 to have no states in common—we can simply rename states if needed, adjusting the transition relation and the state and final states. We can create a finite automaton $M_{r_1+r_2}$ that accepts $[\![r_1 + r_2]\!] = [\![r_1]\!] \cup [\![r_2]\!]$ by putting together M_1 and M_2 and creating a new start state that transitions to both M_1 and M_2. (More precisely, the new start state transitions to every state that the original start states s_1 and s_2 can transition to, with the same label.) Final states are those from M_1 and M_2, with the addition that the new start start will be final if any one of s_1 or s_2 is final. Formally:

$$M_{r_1+r_2} = (Q, \Sigma, \Delta, s, F)$$

where

- $Q = Q_1 \cup Q_2 \cup \{s\}$ where s is a new state not in Q_1 or Q_2

- $\Delta = \Delta_1 \cup \Delta_2 \cup \{\langle s, a, q\rangle \mid \langle s_1, a, q\rangle \in \Delta_1\} \cup \{\langle s, a, q\rangle \mid \langle s_2, a, q\rangle \in \Delta_2\}$

- F is $F_1 \cup F_2 \cup \{s\}$ if $s_1 \in F_1$ or $s_2 \in F_2$, and is $F_1 \cup F_2$ otherwise.

Pictorially:
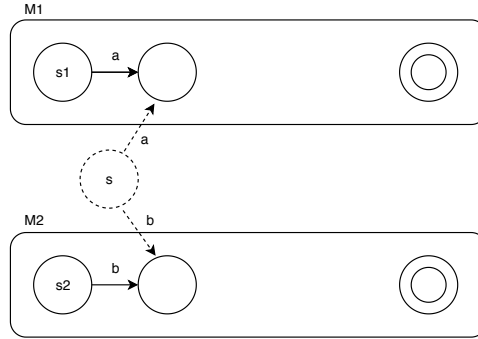
Consider $r_1 r_2$. Again, by recursion we can obtain finite automata $M_1 = (Q_1, \Sigma, \Delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \Delta_2, s_2, F_2)$ that accept the languages of r_1 and r_2 respectively, where Q_1 and Q_2 have no states in common. We can create a finite automaton $M_{r_1 r_2}$ that accepts $[\![r_1 r_2]\!] = [\![r_1]\!] \cdot [\![r_2]\!]$ by putting together M_1 and M_2 and basically sending the final states of M_1 into M_2. (More precisely, every final state of M_1 transitions to every state that the original start states s_2 transitions to, with the same label.) The start state is the original start state s_1 of M_1. Final states are those from M_2, along with the final states of M_1 if the original start state s_2 is final in M_2 (why?). Formally:

$$M_{r_1 r_2} = (Q, \Sigma, \Delta, s_1, F)$$

where

- $Q = Q_1 \cup Q_2$

- $\Delta = \Delta_1 \cup \Delta_2 \cup \{\langle f, a, q \rangle \mid f \in F_1, \langle s_2, a, q \rangle \in \Delta_2\}$

- F is $F_1 \cup F_2$ if $s_2 \in F_2$, and is F_2 otherwise.

Pictorially:
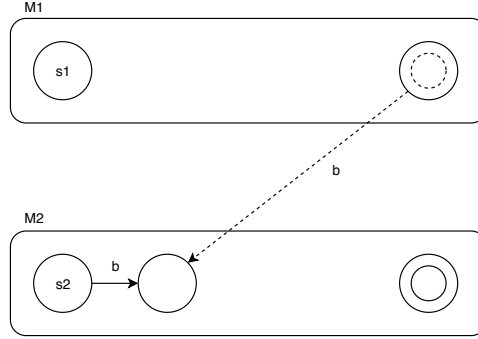
Finally, consider $r_1{}^*$. Again, by recursion we can obtain a finite automaton $M_1 = (Q_1, \Sigma, \Delta_1, s_1, F_1)$ that accepts the language of r_1. We can create a finite automaton $M_{r_1{}^*}$ that accepts $[\![r_1{}^*]\!] = [\![r_1]\!]^*$ by modifying M_1 to send the final states of M_1 back to the start. (More precisely, every final state of M_1 transitions to every state that the original start state s_1 transitions to, with the same label.) The one subtlety is the treatment of the empty string. Since $[\![r_1]\!]^*$ always contains the empty string, we need to make sure the resulting automaton accepts the empty string, which we cannot guarantee by simply taking the original start state of M_1 and making it final—if it wasn't final to start with, and there are transitions back to it, then making it final might make it possible to accept more strings than we want. (Why?) The fix is to create a new start state like we did for $M_{r_1+r_2}$, that transitions to every state that s_1 transitions to, and that is indeed final. Final states are those from M_1, along with the new start state. Formally:

$$M_{r_1{}^*} = (Q, \Sigma, \Delta, s, F_1 \cup \{s\})$$

where

- $Q = Q_1 \cup \{s\}$ where s is a new state not in Q_1

- $\Delta = \Delta_1 \cup \{\langle f, a, q \rangle \mid f \in F_1, \langle s_1, a, q \rangle \in \Delta_1\} \cup \{\langle s, a, q \rangle \mid \langle s_1, a, q \rangle \in \Delta_1\}$.

Pictorially:

13
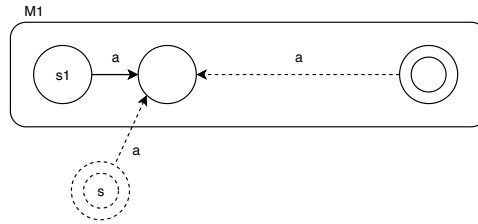
Proving the other direction of the theorem, that the language of a finite automaton is always regular, is a bit more painful. The simplest way is to create a regular expression from a description of the finite automaton, but the construction is awkward. I've added some notes about it below in the supplemental material for this lecture.

Deterministic Finite Automata

In the special case when a finite automaton has, for every symbol of the alphabet, at most one transition out of every state labeled with symbol, we say the finite automaton is *deterministic*. The transition relation Δ for a deterministic finite automaton (DFA) is in fact a partial function (a function that is not defined on all elements of its domain) from $Q \times \Sigma$ to Q. For a DFA, it suffices to start from the start state and follow the one and only path that follows the symbols in the input string. If you don't get stuck on the way and reach a final state after consuming all symbols from the input string, you accept the input string.

(When we want to emphasize that an automaton may not be deterministic, we often use the term nondeterministic.)

Example: Finite automaton M_{even} from earlier is a deterministic finite automaton. Finite automaton M_{last} is a nondeterministic finite automaton.

Even though DFAs seem more restrictive than nondeterministic finite automata, it turns out that DFAs accept exactly the regular languages. Clearly, since every DFA is a finite automaton, if a DFA accepts a language A then A is regular. But if A is regular, while we know it can be accepted by a finite automaton, it's not clear it can be accepted by a *deterministic* finite automaton.

Consider the following way to "execute" a nondeterministic finite automaton M. The idea is to look at the possible transitions that can be taken for a given symbol of the input all at once. Intuitively, we begin in the start state, and for every symbol of the input string in order, we keep track of the possible states we can reach by following a transition labeled with that symbol from any of the possible states we have already reached. Thus, at every step in this process, we maintain a set of possible states that M can be in. If at the end of the string one of the possible states is final, then M accepts the string.

This process, where we consider *sets* of states of M, in fact describes the

execution of a deterministic finite automaton, in which states are sets of states of M, and there is a transition from a set X of states of M to a set Y of states of M labeled a exactly when the transitions of M labeled a from any state in X lead to the states in Y.

Formally, for every finite automaton M, there exists a deterministic finite automaton $\widehat{M}$ that accepts the same language as M, constructed as follows. If $M = (Q, \Sigma, \Delta, s, F)$, construct

$$\widehat{M} = (\widehat{Q}, \Sigma, \widehat{\Delta}, \widehat{s}, \widehat{F})$$

where:

- $\widehat{Q} = \{X \mid X \subseteq Q\}$

- $\widehat{\Delta} = \{\langle X, a, Y \rangle \mid X \subseteq Q, Y = \{q \in Q \mid \langle p, a, q \rangle \in \Delta \text{ for some } p \in X\}\}$

- $\widehat{s} = \{s\}$

- $\widehat{F} = \{X \subseteq Q \mid X \cap F \neq \varnothing\}$

This construction is called the *subset construction*.

We can see that $\widehat{M}$ is deterministic because any subset $X \subseteq Q$ uniquely determines a subset Y to transition to in the definition of $\widehat{\Delta}$.

Supplementary Material

The Language of a Finite Automaton is Regular

We stated in class the following result:

Theorem: *A language A is accepted by some finite automaton M if and only if A is regular.*

We showed one direction of this equivalence: when A is regular, we can construct a finite automaton which accepts A.

To argue the other direction, that if a language is accepted by a finite automaton, then it is regular, we give a process for constructing a regular expression which denotes the same language as that accepted by any given finite automaton.

Let $M = (Q, \Sigma, \Delta, s, F)$ be a finite automaton.

15

We define a recursive function $R(p, q, X)$ that returns a regular expressions representing all strings that take automaton M from state p to state q through transitions going only through states in X, with the possible exception of p and q which need not lie in X.

Here is the recursive function definition. It recurses on the size of X.

When X is empty, let $a_1, \ldots, a_k$ be all the symbols in Σ such that $\langle p, a_i, q \rangle \in \Delta$, that is, that make the automaton transition from p to q. We have two cases: when $p \neq q$,

$$R(p, q, \varnothing) = \begin{cases} a_1 + \cdots + a_k & \text{if } k \geq 1 \\ 0 & \text{if } k = 0 \end{cases}$$

and when $p = q$,

$$R(p, p, \varnothing) = \begin{cases} a_1 + \cdots + a_k + 1 & \text{if } k \geq 0 \\ 1 & \text{if } k = 0. \end{cases}$$

For the recursive case, choose any element $v \in X$. (Different choices will yield different regular expressions at end, but all of those different regular expressions will denote the same language.) Take:

$$R(p, q, X) = R(p, q, X\backslash\{v\}) + R(p, v, X\backslash\{v\})(R(v, v, X\backslash\{v\}))^* R(v, q, X\backslash\{v\})$$

(Set $A\backslash B$ is the set of all elements of A minus all elements of B.) To understand this definition, observe that any string that takes the automaton from p to q with all intermediate states in X either:

- never takes the automaton through state v, and those strings are given by this part of the regular expression:

$$R(p, q, X\backslash\{v\})$$

- or makes the automaton reach state v a first time, given by this part of the regular expression:

$$R(p, v, X\backslash\{v\})$$

followed by a finite number of loops (possibly zero) from v back to itself without going through v in between and always staying in X, give by this part:
$$(R(v, v, X\backslash\{v\}))^*$$

followed by taking the automaton from v to q without going through v, given by this part:
$$R(v, q, X \backslash \{v\}).$$

Once we have such a function R, we can construct the regular expression whose language is the language accepted by M, namely all the strings taking M from the start state to any of the final states $f_1, \ldots, f_k$ through any state of M by:
$$R(s, f_1, Q) + \cdots + R(s, f_k, Q).$$

The result is usually a *huge* unreadable regular expression. Thankfully, it can be simplified because regular expression obey algebraic-like simplification rules. But that's beyond the scope of these notes.

Complementation of Regular Languages

Using finite automata, we can show that the complement of a regular language is regular.

Let A be a regular language. That means that there is a deterministic finite automaton $M = (Q, \Sigma, \Delta, s, F)$ that accepts A. (The determinism is important.)

The first thing we do is *complete* M, in such a way that for every symbol in Σ and for every $q \in Q$, there is a $p \in Q$ such that $\langle q, a, p \rangle \in \Delta$ — in other words, there is a transition at every state for every symbol. We do so by introducing a new *deadend* state to which we transition whenever the original M doesn't have a suitable transition.

Let $M' = (Q', \Sigma, \Delta', s, F)$ by taking

- $Q' = Q \cup \{s_{deadend}\}$ where $s_{deadend}$ is a new state not in Q

- $\Delta' = \Delta \cup \{\langle q, a, s_{deadend} \rangle \mid q \in Q, a \in \Sigma, \text{and there is no } p \in Q \text{ with } \langle q, a, p \rangle \in \Delta\}$

It is not difficult to argue that M' accepts the same language as M. Now, take M', and construct the finite automaton $\overline{M'} = (Q', \Sigma, \Delta', s, Q' \backslash F)$. The language accepted by $\overline{M'}$ is $\overline{A}$, and therefore $\overline{A}$ is regular.

Non-Regular Languages

Finite automata also let us argue that some languages are not regular.

Let $A = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$. That is, A is the language of all strings made up of a sequence of $\mathtt{a}$s followed by a sequence of the same number of $\mathtt{b}$s. I claim A is not regular.

Let's argue by contradiction. Let's assume A *is* in fact regular, and derive a contradiction. This means that our assumption cannot be, and thus A cannot be regular.

If A is regular, then there exists a deterministic finite automaton $M = (Q, \{\mathtt{a}, \mathtt{b}\}, \Delta, s, F)$ that accepts A. Let $N = |Q|$, the number of states in M.

Consider the string $u = \mathtt{a}^{2N}\mathtt{b}^{2N}$, which is a string in A. Therefore M accepts u. Since M has only N states and string u has length $4N$, any sequence of transitions in M that accepts u must hit one state, call it q, at least twice. Moreover, because the first $2N$ symbols of u are all $\mathtt{a}$, that state q must be reached from s after $k < 2N$ transitions labeled $\mathtt{a}$ and also after $k + m < 2N$ transitions labeled $\mathtt{a}$ for some $m > 0$.

In other words, there is a loop of length m from q to q with transitions labeled $\mathtt{a}$. The string $u = \mathtt{a}^{2N}\mathtt{b}^{2N} = \mathtt{a}^k\mathtt{a}^m\mathtt{a}^{2N-k-m}\mathtt{b}^{2N}$ is accepted by the automaton by first going from s to q (via $\mathtt{a}^k$) and then from q to q (via $\mathtt{a}^m$) and then from q to a final state (via $\mathtt{a}^{2N-k-m}\mathtt{b}^{2N}$).

But consider the string $v = \mathtt{a}^k\mathtt{a}^m\mathtt{a}^m\mathtt{a}^{2N-k-m}\mathtt{b}^{2N}$. It also must be accepted by A! Indeed, $\mathtt{a}^k$ takes M from s to q, $\mathtt{a}^m$ takes M from q to q, $\mathtt{a}^m$ takes M from q to q again (second time around the loop!), and finally $\mathtt{a}^{2N-k-m}\mathtt{b}^{2N}$ takes M from q to a final state. So M accepts v. But v is of the form $\mathtt{a}^m\mathtt{a}^{2N}\mathtt{b}^{2N}$ and since $m > 0$, $v \notin A$. So A cannot be the language accepted by M. But we chose M specifically to have language A. That's our contradiction.

Our initial assumption cannot be: A is not regular.

A similar argument shows that $Pal(\Sigma)$, the set of all palindromes over Σ, is not regular.

A general form of the argument is formalized as the *Pumping Lemma for Regular Languages*:

Theorem (Pumping Lemma): *If A is regular language, then there is a number p with the property that any string $u \in A$ of length at least p can be written as $u = xyz$ for x, y, z satisfying:*

  (i) for every $i \geq 0$, $xy^iz \in A$,

  (ii) $|y| > 0$, and

  (iii) $|xy| \leq p$.