Up to this point, what we have covered in the class concerns regular languages exclusively. The main chain of development is that all four varieties of defining mechanisms—DFA, NFA, NFA with ε moves, and regular expressions—are all equivalent in the scope of sets representable by them. And they are called collectively the regular languages.

The proof of the equivalence of all four varieties is carried out by a circular chain of four proofs.

First we proved by the subset construction that any language accepted by an NFA is accepted by a DFA. Here the idea is the following: Since a given NFA has only a finite number of states, the amount of information it keeps track of must be finite. More precisely, since by definition a string is accepted by the NFA iff it leads by some choice of moves to an accepting states, we should keep track of all the states it can possibly reach at all time. Thus, enter the DFA simulating the NFA: namely the states of the DFA are the subsets of the NFA. Transition is accomplished by collecting together all possible moves from the set of NFA states (which is a single DFA state) and the newly arrived DFA state is the set of all NFA states it can reach.

This subset construction can be easily adapted to NFA’s with ε moves. We simply “look forward” in such an NFA with ε moves, to see what states can be reached by any sequence of zero or more ε moves. This is called “taking the ε-closure”. We also make the set of states in the ε-closure of q₀ (the given starting state) as the new starting state in the simulating DFA.

Second we proved that any DFA can be represented by a regular expression (usually known as Kleene’s theorem, a former professor at UW Madison). Here the idea is dynamic programming. While it might be difficult at first glance to capture all strings going from state a to state b, it is rather simple to represent by regular expressions all such strings going from state a to state b without going through any other state at all. Now the clever idea of Kleene: let’s index a family of expressions which capture all strings going from state a to state b without going through any state numbered higher than k. When k is 0, trivial; when k = n, we are done. The recursive formula is $R_{ij}^k = R_{ij}^{k-1} + R_{ik}^{k-1} \cdot (R_{kk}^{k-1})^* \cdot R_{kj}^{k-1}$.

(The reading should be: in order to go from i to j without going through anything higher than k means, without going through anything higher than k − 1, or going to k (the first time, so that without going through anything higher than k − 1), then loop around k indefinitely, and finally taking off to j.) Once this idea is firmly grasped, there should be no need to memorize the formula. With essentially the same idea the book gives the method of doing this by introducing the so-called generalized FA, where transitions of the FA diagram are labeled by regular expressions.

There remains the task of simulating any regular expression by an NFA. (Here is a good example of what I call the “socks” principle: logically we could
have asked to simulate the regular expression by a DFA instead, but since NFAs give us more flexibility in designing the automaton, we use the more flexible tool, namely the NFAs. A similar situation happened in the Kleene’s theorem. We could have asked to start with an arbitrary NFA to be simulated by a regular expression, but since DFAs are more restrictive, we choose to start with a DFA. In general, the “socks” principle says: in order to show A is captured by B, always use the most restrictive (yet general) form of A and use the most flexible form of B. Indeed, to simulate a regular expression by an NFA, we even make the NFA more flexible by introducing ε moves. To simulate a regular expression $R$ by an NFA with ε moves $N$ is by induction on $R$.

As a by-product of the above chain of proofs, we see that the class of regular languages are closed under various operations, such as union (best seen by regular expression), intersection, complement (best seen by DFA), concatenation (best seen by regular expression), *-operation (best seen by regular expression) etc. And they are closed under finitary alternations (add a string, subtract a string, how do you show that?).

Going forward, the main topic now is a general technique for showing certain languages are not regular. It is called the Pumping Lemma:

**Theorem 1** For any regular language $L$, there exists an integer $n$, such that for any string $x \in L$ of length $|x| \geq n$, there must be a decomposition $x = uvw$, where, $|uv| \leq n$, $|v| > 0$, and $uv^iw \in L$ for all $i \geq 0$.

The proof is the “fool the automaton” argument: Suppose $L$ is regular, and hence accepted by a DFA. Say $M$ has $n$ states. We show that after no more than $n$ steps, $M$ must enter a state which has been visited before. This is just a counting argument—also called the “Pigeon Hole principle”—after $n$ steps, there are altogether $n+1$ steps along the way, including the initial position $q_0$. Thus some state $q$ has been visited twice. Let $u$ be the steps before first getting to $q$ and $v$ the loop leading back to $q$, clearly, if we loop it for any number of times (i.e. $v^i$), we will still be in $q$. Thus, $M$ sees no difference after $uv$ and $uv^i$, and thus arrives in the same accepting state after $uv^iw$ as after $x = uvw \in L$.

The use of the Pumping Lemma is usually its contra-positive: Suppose a non-regular language $L$ were regular, and thus it satisfies the Pumping Lemma; we derive a contradiction. A typical example is $L = \{a^n^2 \mid n \geq 0\}$. If $L$ were regular then an infinite subset $\{a^{k+i} \mid i = 0, 1, 2, \ldots\}$ would be in $L$, but successive members of $L$ have larger and larger gaps. This contradiction shows that $L$ is not regular. The same argument can be used to show that $\{a^n^3 \mid n \geq 0\}$ or $\{a^n^4 \mid n \geq 0\}$ . . . , are all non-regular. (How about $\{a^n^5 \mid n \geq 0\}$?)

Try your hand on the following examples: $\{a^n b^n \mid n \geq 0\}$, $\{a^n b^n^3 \mid n \geq 0\}$, $\{a^n b^{n+1} c^{n+2} \mid n \geq 0\}$.

Finally, we will discuss the following important theorem due to Myhill and Nerode. First let me define a few things. We all know what an equivalence relation (a binary relation $R$ which is reflexive, symmetric, and transitive) and its
equivalence classes are (the equivalence class \([x]\) of \(x\) is just all those equivalent to it). An equivalence relation \(R\) is right invariant iff \(xRy\) implies \(xzRyz\) for all \(z\). The index of an equivalence relation is just the number of its equivalence classes.

Given any language \(L\) we can define a particular relation \(R_L\) as \(xR_Ly\) iff for all \(z\), \(xz \in L \iff yz \in L\). Clearly \(R_L\) is an equivalence relation which is right invariant.

**Theorem 2** The following statements are equivalent:

1. \(L\) is regular.
2. The relation \(R_L\) is of finite index.
3. \(L\) is the union of some equivalence classes of a right invariant equivalence relation \(R\) of finite index.

The proof starts by assuming \(L\) is regular and thus accepted by a DFA. We claim that \(\delta(q_0, x) = \delta(q_0, y)\) implies \(xR_Ly\). This is clear, since just as in the proof of the Pumping Lemma, after \(x\) and \(y\) reaches the same state, they are inseparable thereafter! So each state of the DFA serves as a name. Every string \(x\) must be equivalent under \(R_L\) to one of them, namely the state it ends up with. But there are at most \(n\) states in the DFA, so that \(R_L\) is of finite index.

Given that \(R_L\) is of finite index, then let’s collect those equivalence classes of \(R_L\) which contain members in \(L\) (by the definition of \(R_L\), if one member belongs to \(L\), then all members of the same class belong to \(L\).) Now just take the union of all these classes, which must equal \(L\).

Now suppose we are given \(L\) as the union of some equivalence classes of a right invariant equivalence relation \(R\) of finite index. We define a DFA out of the equivalence classes of \(R\). The states are the equivalence classes \([x]\), which is finite in number. \(\delta([x], a) = [xa]\), this is well-defined (i.e., independent of the choice of \(x\) as the name for \([x]\) since \(R\) is right invariant.) We choose \([\epsilon]\) to be our starting state (why?), and what is your choice for the final states?

This completes the proof. Very importantly the proof establishes the following: given any DFA for a regular set \(L\), \(R_L\) can be obtained by collapsing possibly some states of the DFA. Then the equivalence classes of \(R_L\) can be used to design a “reduced” DFA accepting \(L\), which can be viewed as obtained by collapsing some states in the original DFA. Thus, every regular set \(L\) has a unique minimum state machine and every DFA for \(L\) can be reduced to it.