1. The finite-state automata (FSAs) that we have looked at thus far are deterministic finite-state automata (DFSAs): there is never more than one transition possible given a state and the current position in the string.

2. In non-deterministic FSAs (NFSAs), there can be more than one transition possible given a state and the current position in the string.

3. Two sources of non-determinism:
   (a) The transition relation, when applied to a given pairing of a state $q$ and input $i$, may give more than one resulting state.
   (b) In NFSAs, the empty string $\epsilon$ is a possible input to the transition relation. Transitions involving empty input are called $\epsilon$-transitions.

4. So, an NFSA is defined as follows:
   - A finite set of $N$ states $Q = \{q_0, q_1, \ldots, q_{N-1}\}$, with $q_0$ the start state
   - A finite input alphabet $\Sigma$ of symbols (the symbols that comprise strings, like in regexes)
   - A set of final states $F \subseteq Q$
   - A transition relation between states. The transition relation $\delta(q, i)$ takes two arguments—a state $q \in Q$ and an input symbol $i \in \Sigma \cup \{\epsilon\}$—and returns a set of possible new states $Q' \subseteq Q$.

5. Let’s look at some DFSA and NFSA pairs that are equivalent: they accept the same strings and reject the same strings as each other.
6. What are the regular expressions that correspond to these two automata?

7. **Recognizing strings with an NFSA:** the question is, does a given FSA $a$ recognize a given string $s$? For DFSAs, the algorithm was extremely simple. But NFSAs are more complicated, because there are choicepoints! Options:

   - **Backup:** whenever we encounter a choicepoint, generate a list $l$ of options in $a$ and mark our position in $l$ and $s$. Whenever we fail, go back to the last choicepoint and try the next option; if we run out of options, then the string is rejected.
   
   - **Lookahead:** look forward in the string to guide our choice of options. Look ahead in the textbook to learn about lookahead!
   
   - **Parallelism:** Instead of maintaining and updating a single state, maintain the set of possible states.

8. **Example of backup:** Consider recognition of **to** with the automaton below:

9. **Example of parallelism:** consider recognition of **cabi** with automaton (2)
10. **Equivalence of NFSAs and DFSAs:** We saw examples of equivalent NFSAs and DFSAs. It turns out that any NFSA has an equivalent DFSA, and there is an algorithm that will determinize any NFSA. NFSAs are easier to write than DFSAs! For any NFSA, the equivalent DFSA generally has more states, but the recognition algorithm for DFSAs is simpler.

11. **Regular languages:** any language definable by a regex or an FSA is a regular language. We can also define the set of regular languages over a symbol set $\Sigma$ inductively:

- $\emptyset$ is a regular language
- For all $a \in \Sigma \cup \epsilon$, $\{a\}$ is a regular language
- If $L_1$ and $L_2$ are regular languages, then so are their concatenation, disjunction, and Kleene closure

12. This means that regular languages have beautiful closure properties. For any regular language $L_1$, its complement and reversal are both regular languages. Furthermore, for any two regular languages $L_1$ and $L_2$, their intersection, union, and difference are also regular languages.

13. We’re now ready to show how to construct an FSA from any regular expression $r$! Here are the base cases:

\[
\begin{align*}
&\text{start} \rightarrow q_0 \quad r = \epsilon \\
&\text{start} \rightarrow q_0 \quad r = \emptyset \\
&\text{start} \rightarrow q_0 \quad r = a
\end{align*}
\]

And here are the additional operators we need for inductive construction from automata $A$ and $B$:

- **Concatenation**

\[
\begin{align*}
&\text{start} \rightarrow q_0 \quad \epsilon \\
&\text{start} \rightarrow q_F \quad \epsilon
\end{align*}
\]

- **Disjunction**

\[
\begin{align*}
&\text{start} \rightarrow q_0 \quad \epsilon \\
&\text{start} \rightarrow q_0 \quad \epsilon
\end{align*}
\]

- **Kleene closure**

\[
\begin{align*}
&\text{start} \rightarrow q_0 \\
&\text{start} \rightarrow q_F \quad \epsilon
\end{align*}
\]

where arrows into an automaton are always into the automaton’s start state, and appropriate changes of final-state status are made.