The formal definition of a finite automaton is a 5-tuple (i.e., an object) $N = (Q, \Sigma, \delta, s, F)$ where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet
- $s$, a member of $Q$, is the start state (also called $q_0$)
- $F$, a subset of $Q$, is the set of accepting states (also called final states)
- $\delta$ is a finite set of instructions (also called transitions) of the form $(p, c, q)$ where $p, q \in Q$ and $c \in \Sigma$; an NFA with $\varepsilon$-transitions (NFA$_\varepsilon$) also allows $(p, \varepsilon, q)$.

The machine is deterministic (a DFA) if $(\forall p \in Q)(\forall c \in \Sigma)(\exists ! q \in Q): (p, c, q) \in \delta$. Else it is "properly" nondeterministic (an NFA).

So DFA is a special case of an NFA. When we have a DFA $M$, we can regard $\delta$ as a function from $Q \times \Sigma$ to $Q$.

With an NFA, we could regard $\delta$ as a function from $Q \times \Sigma$ to $2^Q$, which is the set of all subsets of $Q$ and called the power set of $Q$. But in most cases I prefer to think of $\delta$ as a set of instructions---the same as "trominoes" in my previous lecture.

Say that $N$ can process a string $x$ from state $p$ to state $q$ if there is a sequence of instructions

$$(p, c_1, q_1)(q_1, c_2, q_2)(q_2, c_3, q_3) \cdots (q_{m-2}, c_{m-1}, q_{m-1})(q_{m-1}, c_m, q)$$

such that $c_1c_2 \cdots c_m = x$. Then we write $x \in L_{pq}$ (with $N$ understood). Now formally define:

$$L(N) = \cup_{f \in F} L_{sf}.$$  

If $N$ has only one accepting state (a design goal we can meet for NFAs but often not for DFAs) then the language is just $L_{sf}$. We will find the $L_{pq}$ concept especially handy with "GNFAs" on Fri.

Without the dead state and arc to it, the NFA $N$ on input $x = $DD would "crash" in state $s$. Even though $s$ is an accepting state (and even though this would count as legal termination by a Turing machine), not all of $x$ would be processed, so it does not count in the FA's language.

With the dead state present, $x$ gets processed to $\text{dead}$, but $\text{dead} \notin F$ so $x \notin L(N)$ still.
Regular Expressions and Their Corresponding NFAs (with \(\epsilon\)-transitions):

**(B1)\(\emptyset\) is a regexp; \(L(\emptyset) = \emptyset;\)  \(N_{\emptyset} = \) \[\text{Diagram of } N_{\emptyset}\]  \((\delta = \emptyset)\)

**(B2)\(\epsilon\) is a regexp; \(L(\epsilon) = \{\epsilon\};\)  \(N_{\epsilon} = \) \[\text{Diagram of } N_{\epsilon}\]  \(\delta\) has \((s, \epsilon, f)\)

For all chars \(c \in \Sigma:\)

**(B3)\(c\) is a regexp; \(L(c) = \{c\};\)  \(N_c = \) \[\text{Diagram of } N_c\]  \(\delta\) has \((s, c, f)\)

This completes the basis of an inductive definition of regular expressions. Now let \(\alpha\) and \(\beta\) be any two regular expressions, with languages \(A = L(\alpha)\) and \(B = L(\beta)\). By inductive hypothesis (IH) we have NFAs \(N_\alpha\) and \(N_\beta\) such that \(L(N_\alpha) = A\) and \(L(N_\beta) = B\). Then:

**\((I1)\)\(\gamma = \alpha + \beta\) is a regexp; \(L(\gamma) = A \cup B.\)**

Now to complete the induction case \((I1)\) we need to show how to build an NFA \(N_\gamma\) such that \(L(N_\gamma) = L(\gamma)\). What we have to work with is (are) \(N_\alpha\) and \(N_\beta\). We know they have start states we can call \(s_\alpha\) and \(s_\beta\). Taking a cue from the base case NFAs, and mainly for convenience, we may suppose they have unique accepting states \(f_\alpha\) and \(f_\beta\). Besides that, we make no assumptions about their internal structure, so we draw them as "blobs":

\[\text{Diagram of } N_\gamma\]

The goal is to connect them together to make \(N_\gamma\) with needed properties, also for the cases:

**\((I2)\)\(\gamma = \alpha \cdot \beta\) is a regexp; \(L(\gamma) = A \cdot B.\)**

**\((I3)\)\(\gamma = \alpha^*\) is a regexp; \(L(\gamma) = A^*.\)**  \((\text{In } I3 \text{ we have only } N_\alpha \text{ given.})\)
The proof will be finished with the star case (I3) on Wednesday.

NFAs all having a single accepting state.

complete the proof. But first, are you completely happy with [I will continue as time permits by copy-and-paste and moving things around to do the other two inductive cases to complete the proof. But first, are you completely happy with \( N_\gamma \) as it stands?]

[Answer was no: adding the state \( f_\gamma \) and \( \epsilon \)-arcs shown in red "preserves the invariant" of the NFAs all having a single accepting state.]

Construction for (I1):

\[
\begin{array}{c}
N_\gamma = \\
\begin{array}{c}
 s_\alpha \quad N_\alpha \quad f_\alpha \\
 \quad \quad \quad \quad \epsilon \\
 s_\beta \quad N_\beta \quad f_\beta \\
 \quad \quad \quad \quad \epsilon
\end{array}
\end{array}
\]

This builds \( N_\gamma \), but we still need to prove it is correct, i.e., \( L(N_\gamma) = L(\gamma) \). \textbf{Note the rhythm:}

1. \( L(N_\gamma) = L(N_\alpha) \cup L(N_\beta) \) by machine construction;
2. \( L(N_\alpha) = L(\alpha) \) and \( L(N_\beta) = L(\beta) \) by inductive hypothesis;
3. Thus \( L(N_\gamma) = L(\alpha) \cup L(\beta) = L(\alpha + \beta) = L(\gamma) \) by definition of \( \gamma \).

[I will continue as time permits by copy-and-paste and moving things around to do the other two inductive cases to complete the proof. But first, are you completely happy with \( N_\gamma \) as it stands?]

[Answer was no: adding the state \( f_\gamma \) and \( \epsilon \)-arcs shown in red "preserves the invariant" of the NFAs all having a single accepting state.]

(12) \( \gamma = \alpha \cdot \beta \) is a regexp; \( L(\gamma) = A \cdot B = \{xy : x \in A \land y \in B\} \).

\[
\begin{array}{c}
N_\gamma: \\
\begin{array}{c}
 s_\alpha \quad N_\alpha \quad f_\alpha \\
 \quad \quad \quad \quad \epsilon \\
 s_\beta \quad N_\beta \quad f_\beta
\end{array}
\end{array}
\]

Then \( L(N_\gamma) = L(N_\alpha) \cdot L(N_\beta) \) because....processing....

To write the reasoning out: \( N_\gamma \) can process a string \( z \) from its start state \( s_\gamma = s_\alpha \) to its (unique) final state \( f_\gamma = f_\beta \) if and only if \( z \) has a first part \( x \) that gets processed from \( s_\alpha \) to \( f_\alpha \) and a second part \( y \) that gets processed from \( s_\beta \) to \( f_\beta \) (with the \( \epsilon \) from \( f_\alpha \) to \( s_\beta \) silently in-between). I.e.:

\[
z \in L(N_\gamma) \iff z \in \{x \cdot y : x \in L(N_\alpha) \land y \in L(N_\beta)\} \iff z \in L(N_\alpha) \cdot L(N_\beta).
\]

Thus \( L(N_\gamma) = L(N_\alpha) \cdot L(N_\beta) \). By \textbf{IH}, this equals \( L(\alpha) \cdot L(\beta) \), which by how the semantics of \( \gamma = \alpha \cdot \beta \) is defined via \( L(\gamma) = L(\alpha) \cdot L(\beta) \) finally gives us the needed conclusion \( L(N_\gamma) = L(\gamma) \).

The proof will be finished with the star case (I3) on Wednesday.