In this lecture we will formalize the equivalence between regular expressions and regular languages. To do so, we first need to formalize the notion of a regular expression. So far we have only been performing regular operations on languages, but haven’t defined the base case. The three base cases are single character, $\epsilon$, and $\emptyset$. Then a regular expression is formally one of:

1. $a$ for some $a \in \Sigma$. 
2. $\epsilon$, the empty string. 
3. $\emptyset$, the empty language. Note that this is not the same as the empty string. 
4. $(R_1 \cup R_2)$ where $R_1$ and $R_2$ are regular expressions. 
5. $(R_1 \circ R_2)$ where $R_1$ and $R_2$ are regular expressions. 
6. $(R_1^\ast)$ where $R_1$ is a regular expression.

In the first three cases, the corresponding languages are $\{a\}$, $\{\epsilon\}$, and $\emptyset$ respectively. In cases 4 - 6, which are inductive, the languages are formed by performing regular operations on the languages corresponding to $R_1$ and $R_2$ (if $R_2$ is needed).
1 Languages Described by Regular Expressions are Regular

The plan is to show that every language described by a regular expression can be described by an NFA.

The equivalence between NFA and DFAs then gives that such languages are also describable by DFAs, and are thus regular.

The overall proof is by induction on the length of the regular expression. Note that the recursive operations $\cup$, $\circ$ and $\ast$ all put together shorter expressions. So we can assume that those expressions already correspond to regular languages, and in turn have NFAs that accept them.

The base cases are the three base cases for regular expressions. For these we give explicit constructions:

1. $a$: two states, with one transition from the start to the accepting state corresponding to $a$.

2. $\epsilon$: one starting state that’s also accepting.

3. $\emptyset$: one starting state that’s not accepting.

The inductive case is similar to how we applied regular operations to regular languages. We will work with $N_1$ and $N_2$, the NFAs accepting $L_1$ and $L_2$ respectively.

1. $\cup$: a new ‘super starting state’ with $\epsilon$ transitions to both starting states of $M_1$ and $M_2$.

2. $\circ$: start at $N_1$’s starting state, and add $\epsilon$ transitions from each accepting state of $L_1$ to the starting state of $N_2$.

3. $x^\ast$: we need to have an ‘extra’ starting state to accept $\epsilon$. For this we create an extra starting state that’s accepting, and has an $\epsilon$ transition to the starting state of $N_1$.

The inductive nature of this proof means we can (and also need to) compose these constructions together. We will work through the example (1.58) in the text book of $(a \cup b)^\ast aba$.

2 Regular Expressions for Regular Languages

It remains to show the other direction of this equivalence. That the language accepted by any DFA is a regular language.

For this, we will gradually shrink a DFA. However, we will allow for regular expressions on the transitions, instead of just a single character from the alphabet. This is knowns a generalized nondeterministic finite automata (GNFA).

For simplicity we assume that a GNFA has:
1. No edges entering its starting state. This can be done by creating a duplicate of the starting state, and direct transitions to the starting state to these instead.

2. Only one accepting state. This can be done by adding $\epsilon$ transitions from

3. At most one edge between any pair of states, and at most one self-loop per state. These can be done by removing duplicate edges via the $\cup$ operation: we simply union the regular expressions on parallel edges.

Note that a GNFA with no intermediate states, just start and accept, is precisely a regular expression: only those strings matched by expressions on the one edge from start to accept is accepted.

Our plan is to inductively move to this state, with the key operation being reducing one intermediate state. Consider some intermediate state $q_{rip}$. Suppose it’s visited at some point in the NFA, with the previous state being $q_1$, successor being $q_2$. Then let the expressions on the arrows be:

1. $R_1$ for going from $q_1$ to $q_{rip}$.
2. $R_2$ for going from $q_{rip}$ to $q_2$.
3. $R$ for going from $q_{rip}$ to itself.

Then the only sequence of characters matched is first $R_1$, then some number of $R$, then $R_2$. So we can instead add an arrow from $q_1$ to $q_2$ with label

$$R_1 \circ (R^*) \circ R_2.$$ 

Repeating this for every $q_1$ with arrow to $q_{rip}$, and every $q_2$ that $q_{rip}$ has an arrow to then covers all possible ways of passing through $q_{rip}$, and thus removes the need to go through $q_{rip}$. 
