DISCLAIMER: These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

Today’s topics:

1. Building NFAs from Regular Expressions.
2. Building Regular Expressions from NFAs (GNFAs)

1 Regular Operations Keep Languages Regular

We will show that the class of regular languages is precisely the languages composable from regular operations using the elements from $\Sigma$.

As a sanity check, a single string can be written as the concatenation of its characters. For example

$$\{abc\} = \{a\} \circ \{b\} \circ \{c\},$$

and a set of strings is simply the union of languages corresponding to single strings.

Formally, we want to verify that the union/concatenation of two regular languages, or the star of a regular language is also regular. For concatenation / star, a major challenge is figuring out, or guessing, where one word ends and the next begins.

Concretely, we will show that every language described by a regular expression can be described by an NFA.

The equivalence between NFA and DFAs (see e.g. Theorem 1.39 and Example 1.41 of textbook) 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 $*$ 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. $x^*$: 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 $(a \cup b)^*aba$. Example 1.56 is also worth a look as well.

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.

To see this intuitively, consider the 2-state DFA shown in Example 1.68 of the textbook. It can start with an arbitrary number of $a$s, then takes a $b$, and then any combination of $a$ and $b$ keeps it in the finishing state. So the regular expression describing the language accepted by this DFA is

$$a^* \circ b \circ (a \cup b)^* .$$

For the general case, we will gradually shrink a DFA and extract out the regular expression from the final version with two states. 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 $q_{rip}$.

### 3 Pumping Lemma

Then the question is how to show some language is not regular. Here we can turn to the pumping lemma.

**Theorem 3.1** (Pumping Lemma for Regular Languages). If $A$ is a regular language, then there is a number $p$ (the pumping length) where, if $s$ is a string in $A$ and $|s| \geq p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

We can use this lemma to show that the language

$$B = \{0^n1^n \mid n \geq 0\}$$

is not regular.

Assume for the sake of contradiction, that $B$ is regular. Then, let $p$ be the pumping length given by the lemma. Pick a string $s = 0^p1^p$. Since $s \in B$ and $|s| \geq p$, the pumping lemma implies that $s$ can be divided into three pieces $s = xyz$, such that $xy^iz \in B$ for all $i \geq 0$. Let us consider the ways in which $s$ cannot be divided into $x$, $y$ and $z$:
1. Suppose \( y \) has only 0s. Then \( xy^2z \) has more 0s than 1s (note that it cannot be length 0 from condition 2 of the lemma), so it cannot be in \( B \).

2. The case where \( y \) has only 1s follows similarly.

3. Suppose \( y \) has both 0s and 1s. Then, in \( xy^2z \), some 1s appear before 0s and so it is not in \( B \).

Hence, there is no division of \( s \) which satisfies the properties of the lemma. This must mean that our initial assumption that \( B \) is regular was false.

**Things to note**

- You **cannot** choose \( p \) - you have to assume this is given by the lemma.
- You **can** choose the string \( s \).
- You **cannot** choose one particular division of \( s \). You have to prove that no possible division can satisfy the three conditions.

### 4 Proof of Pumping Lemma

We applied the lemma as a black box, but let us see why it is true.

*Proof of Theorem 3.1.* Since \( A \) is regular, we can construct a DFA \( M \) recognizing it. Let this be \( M = (Q, \Sigma, \delta, q_1, F) \). Let us assign the pumping length to be \( p = |Q| + 1 \). Now, let \( s \) be any string accepted by \( M \) of length at least \( p \). (What if there are no such strings? Then the lemma is true by default!)

Consider the set of states visited by \( M \) on input \( s \). Since \( s \) has at least \( |Q| + 1 \) characters, at least one state must be visited twice. Let \( q_i \) be the first such repeated state. The sequences of states visited look like:

\[
q_1, \ldots, q_i, \ldots, q_i, \ldots, q_F
\]

Let \( x \) denote the first part of \( s \) which the machine reads before reaching \( q_i \) the first time. Let \( y \) denote the part of \( s \) reads between \( q_i \) and returning to \( q_i \). Let \( z \) be the rest of the string. Now, it is easy to see that replacing \( y \) with \( y^i \) for any \( i \geq 0 \) will also lead the machine to reach \( q_F \) (condition 1). \( y \) cannot be empty since is is between two separate occurrences of a state (condition 2). Lastly, since \( q_i \) is the first repetition of a state, the number of characters read until then is at most \( |Q| + 1 = p \) (condition 3).

**Note:** The converse of the pumping lemma is not true! That is, a language satisfying the lemma may still be non-regular.
5 Examples of Using Pumping Lemma

Example 1: pumping up
Let $C$ be the language \( \{1^{n^2} \mid n \geq 0 \} \). $C$ is a unary language, which only accepts strings whose lengths are perfect squares. Let’s use the pumping lemma to prove that $C$ is not regular.

Assume for the sake of contradiction, that $C$ is regular. Then, let $p$ be the pumping length given by the lemma. Pick a string $s = 1^{p^2}$. Since $s \in C$ and $|s| \geq p$, the pumping lemma implies that $s$ can be divided into three pieces $s = xyz$, such that $xy^iz \in D$ for all $i \geq 0$. So, we have that $|xy^iz|$ is always a perfect square.

From condition 3, we know that $|xy| \leq p \implies |y| \leq p$. Then, $|xy^2z| = |xyz| + |y| \leq p^2 + p < (p + 1)^2$. Hence, the only way $|xy^2z|$ can be a perfect square is if $|y| = 0$, which contradicts condition 2 of the lemma.

Example 2: pumping down
Let $D$ be the language \( \{0^i1^j \mid i > j \} \). Let’s use the pumping lemma to prove that $D$ is not regular.

Assume for the sake of contradiction, that $D$ is regular. Then, let $p$ be the pumping length given by the lemma. Pick a string $s = 0^{p+1}1^p$. Since $s \in C$ and $|s| \geq p$, the pumping lemma implies that $s$ can be divided into three pieces $s = xyz$, such that $xy^iz \in D$ for all $i \geq 0$.

From condition 3, we know that $|xy| \leq p \implies y$ has only 0s. Then, $xy^0z = xz$ has equal or lesser 0s than 1s, which contradicts condition 1 of the lemma.