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.

- Homework 1 has been posted.
- Comments from last time: mostly JR, 4 WTS, 6 STS, 3 STF, 1 WTF.
  1. Set theoretic notations confusing.
  2. $F$ was referred to as ‘final states’: it should be accepting states. Why multiple accepting states?
  3. Definitions of automata.
  4. More diagrams / examples: will do today.
- Today’s topics:
  1. Automaton: states, alphabet, transition.
  2. Start state, accept state.
  3. Language: strings accepted by an automata.

Last time we formally introduced the notion of finite state machine, or finite automaton. It’s a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

1. $Q$ is a finite set called the STATES, denoted as blobs.
2. $\Sigma$ is a finite set (corresponding to the input) called the alphabet.
3. $\sigma : Q \times \Sigma \rightarrow Q$ is the transition function. This is denoted as arrows between the blobs, marked by the corresponding alphabet.
4. $q_0 \in Q$ is the starting state, denoted by an arrow entering it from nowhere.
5. $F \subseteq Q$ is the set of accept states, denoted by a double circle.
As a refresher, the text book introduces automatas via an example with three states, \( Q = \{q_1, q_2, q_3\} \), and a binary alphabet \( \Sigma = \{0, 1\} \). The starting state is \( q_1 \), and there is only one accepting state \( q_2 \). The transitions are:

\[
\begin{array}{c|cc}
Q \setminus \Sigma & 0 & 1 \\
q_1 & q_1 & q_2 \\
q_2 & q_3 & q_2 \\
q_3 & q_2 & q_2 \\
\end{array}
\]

We can check that this machine accepts 1101, as well as 1, 01, 0101010101, but does not accept the strings 0, 10, 1010000.

With some careful reasoning, we can check that for this machine \( M \), we have

\[
L(M) = \{ w : w \text{ contains at least one } 1, \text{ and the last } 1 \text{ is followed by an even number of } 0s \}.
\]

There are two ‘corner cases’ worth mentioning:

1. We use \( \epsilon \) (epsilon) to denote the empty string. It’s possible for a machine \( M \) to have \( \epsilon \in L(M) \): we just need \( q_0 \in F \).

2. It’s also possible for \( M \) to accept no strings. For example, \( F \) can simply be empty. In such cases, we formally have \( L(M) = \emptyset \).

### 1 Binary Strings Divisible by 3

Note that binary strings also represent numbers: the leftmost character is the most significant, the rightmost is least significant.

We will use \( \overline{w} \) to represent the number corresponding to the string/word \( w \).

We can create an automata using the following observation: if we attach a character \( x \) onto the end of a word \( w \), we get

\[
\overline{wx} = 2\overline{w} + x.
\]

That is, the remainder of \( \overline{wx} \) when divided by 3 is determined by the remainder of \( \overline{w} \) divided by 3, along with the value of \( x \). Note that the operations (multiplication and addition) are actually manipulating numbers, instead of strings. This is because \( \overline{w} \) and \( \overline{wx} \) are already numbers.

Specifically, we get the following table:

<table>
<thead>
<tr>
<th>( \overline{w} ) mod 3</th>
<th>( \overline{w0} ) mod 3</th>
<th>( \overline{w1} ) mod 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
This is exactly the property we want from an automata: the state of the new word is given precisely by the state of the previous word, and the new character added. We create states

\[ Q = \{ q_0, q_1, q_2 \} \]

to correspond to the case where the string read so far having residue 0, 1, and 2 modulo 3 respectively. Then

- because the empty string corresponds to 0, we start at \( q_0 \), and
- because we want to only accept the strings that correspond to numbers with residue 0 modulo 3, we set \( F = \{ q_0 \} \).

The transitions are almost identical to the table above, except we replace explicit modulos with states

\[
\begin{array}{c|cc}
Q \backslash \Sigma & 0 & 1 \\
\hline
q_0 & q_0 & q_1 \\
q_1 & q_2 & q_0 \\
q_2 & q_1 & q_2 \\
\end{array}
\]

### 2 From (Sets of) Strings to Automatas

We can also go the other way: we can go from any set of strings to an automata that accepts them. For example, suppose we’re in the binary alphabet \( \Sigma = \{ 0, 1 \} \) again, and have

\[ A = \{ 0, 001 \} , \]

then an automata that works is one with five states \( Q = \{ q_1, q_2, q_3, q_4, bad \} \), where we start at \( q_1 \), and have transitions:

\[
\begin{array}{c|cc}
Q \backslash \Sigma & 0 & 1 \\
\hline
q_1 & q_2 & bad \\
q_2 & q_3 & bad \\
q_3 & bad & q_4 \\
q_4 & bad & bad \\
bad & bad & bad \\
\end{array}
\]

that is, we create a new state whenever we’re still in the prefix of one of the strings in \( A \), and accept when we hit exactly one of these strings. That is, we need to start at \( q_1 \), and have

\[ F = \{ q_2, q_4 \} . \]

Formally, this type of logic shows

**Lemma 2.1.** For any finite set \( A \), there is a finite automaton \( M \) such that

\[ L(M) = A. \]
Recall we also defined a regular language to be one that’s accepted by some finite automaton.

**Definition 2.2.** A language is regular if some finite automaton recognizes it.

Note that the lemma that we proved above immediately implies that all finite languages are regular. However, the most interesting languages tend to be the infinite sized ones. In the rest of today’s lecture we will see two examples.