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: WTS: 5, STS: 9, JR: 25, STF: 1, WTF: 1.
  1. Hard to see subscripts on the board.
  2. \( L(M) \) and the notion of a regular language.
  3. Representing transition functions: either arrows or tables are fine.
  4. Transitions with 2 inputs: view it as transiting on the two input characters, one after another.
  5. Why \( 10 = 2 \) and \( 101 = 5 \), why are strings numbers, and why \((\text{mod } 3)\)?
  6. (2x) do you have a pet?
  7. Why regular languages?

- Today’s topics:
  1. Automaton for string matching.
  2. Regular operations on languages.
  3. Quiz 1.
  4. Closure of regular languages under union.

We start off from where we left off last time. That is, we want to design an automaton that (only) accepts the string that contain the some ‘key’ substring.

Consider the alphabet \( \Sigma = \{a, b\} \), and we want to find strings that contain \( ab \) as a substring.

We create states for whether we matched 0, 1, or 2 characters so far. At each state, we move onto the next state when the character matches, and start over (back to \( q_0 \)) if it doesn’t match.

There are two major caveats:

1. After we have encountered \( ab \), we stay in that state no matter what comes afterwards.
2. If we get a

This leads to states $q_0$, $q_1$, $q_2$, with $q_0$ as the starting state, and $q_2$ as the (only) accepting state.

The transitions are

<table>
<thead>
<tr>
<th>$Q \setminus \Sigma$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_1$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

1 Regular Operations

Note that the strings that contain $ab$ can be written as the concatenation of:

1. any arbitrary string,
2. followed by the string $ab$,
3. followed by any arbitrary string.

Each of these building blocks is much easier to build an automaton for.

This then brings us to our next point, which is to decompose regular languages into simpler building blocks. We want to give a way to build a new regular language from two existing ones, $A$, and $B$. The operations that we devise are akin to addition/multiplication /powering.

1. Union: $A \cup B = \{ x : x \in A \text{ or } x \in B \}$.
2. Concatenation: $A \circ B = \{ xy | x \in A \text{ and } y \in B \}$
3. Star: $A^* = \{ x_1 x_2 \ldots x_k | k \geq 0 \text{ and } x_i \in A \forall 1 \leq i \leq k \}$.

In particular, the language described above, $L$ that contains $ab$ as a substring, can be written as

$$\{a, b\}^* \circ \{ab\} \circ \{a, b\}^*.$$  

Note that the $^*$, or star operation, has priority over the $\circ$ operation.

Furthermore, the $^*$ operation denotes all strings (with possibly 0 length) built from things in this set. Here because we started with $\{a, b\}$, it’s the entire set of strings.

A more interesting example is

$$\{a, ab\}^* = \{ \epsilon, a, aa, ab, aab, aba, aaaa, aaab, aaba, abaa, abab \ldots \}.$$  

This is the basis of how we create infinite sized languages: recall that
2 Regular Languages and Regular Operations

We will eventually 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.