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.

- Comments from last time: 2 WTS, 4 STS, 13 JR, 4 STF, 1 WTF.
  1. \( \epsilon \)-transitions / multiple transitions: need examples.
  2. Do the proofs slower.

- Today’s topics:
  1. Non-deterministic operations.
  2. Simulating NFAs.
  3. Examples of NFAs.
  4. Nondeterministic transitions for union, concatenation, and star.

Last time, we introduced the notion of a non-deterministic automata. Here the generalization is:

1. There can be multiple exiting arrows for the same alphabet character at a state.

2. Arrows can also be labeled with \( \epsilon \), which means it does not consume a character of the input, and makes the move.

As an example, consider an automata for accepting strings ending with 01. This is the concatenation of the language of all strings, along with the language containing just 11:

\[
\left( \{0,1\}^* \right) \circ \{11\}
\]

A DFA \( M_1 \) for \( \{0,1\}^* \) can just has one state, \( q_0 \), which is accepting, and both transitions going back to it.

A DFA \( M_2 \) for \( \{11\} \) has four states, \( q_1, q_2, q_3 \), and \( bad = q_4 \). \( q_3 \) is accepting, and the transitions are:

<table>
<thead>
<tr>
<th>( Q )</th>
<th>( \Sigma )</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1 )</td>
<td>( q_2 )</td>
<td>( q_4 )</td>
<td></td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_4 )</td>
<td>( q_3 )</td>
<td></td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4 )</td>
<td>( q_4 )</td>
<td></td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_4 )</td>
<td>( q_4 )</td>
<td></td>
</tr>
</tbody>
</table>
To concatenate these, we simply add an $\epsilon$ transition from $q_0$ to $q_1$: that is, we can switch over from $M_1$ to $M_2$ at an point.

It’s useful to consider how this NFA operates on the input 011101: it essentially ‘branches’ into multiple states at each point of the input.

1 Regular Operations with Non-Determinism

The three regular operations that we discussed can also be applied to languages recognized by non-deterministic automatas (NFAs). In each case, the change that we need to make is significantly simpler:

1. $A_1 \cup A_2$: a new ‘global’ starting state $q_0$ with $\epsilon$-transitions to $q_1$, the starting state for $M_1$, as well as $q_2$, the starting state for $M_2$.

2. $A_1 \circ A_2$: an $\epsilon$ transition from each accepting state of $A_1$, $F_1$, to the starting state of $M_2$, $q_2$.

3. $A^*$: an $\epsilon$ transition from each accepting state in $F$ to $q_0$, the starting state.

Thus, the langauges recognized by NFAs are closed under regular operations. We will show in the next class that this is precisely the regular languages (languages recognized by DFAs).

2 Formal definition of an NFA

We formally define an NFA as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set of states.
2. $\Sigma$ is a finite alphabet.
3. $\delta : Q \times (\Sigma \cup \epsilon) \mapsto \mathcal{P}(Q)$ is the transition function ($\mathcal{P}(Q)$ is the power-set of $Q$).
4. $q_0 \in Q$ is the starting state.
5. $F \subseteq Q$ is the set of accept states.

Note that the only difference from a DFA is in $\sigma$. The first difference is that the transitions can accept $\epsilon$. We sometimes refer to the $\epsilon$-inclusive alphabet as $\Sigma_\epsilon$. Secondly, each transition can now lead to multiple states.

We say that an NFA $N = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w$ if

- we can write $w$ as $w = y_1y_2 \cdots y_m$, where each $y_i \in \Sigma_\epsilon$;
- there exists a sequence of states $r_0r_1 \cdots r_m$ such that
3 Examples

Example 1. Consider the NFA $N_1$:

Formal definition: $N_1 = (Q, \Sigma, \delta, q_0, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$
2. $\Sigma = \{0, 1\}$
3. $\delta =

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>${q_1}$</td>
<td>${q_1, q_2}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${q_3}$</td>
<td>$\emptyset$</td>
<td>${q_3}$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$\emptyset$</td>
<td>${q_4}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>${q_4}$</td>
<td>${q_4}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

4. $q_0 = q_1$
5. $F = \{q_4\}$

Question: What is $L(N_1)$?

First of all let us check if a string, say, 11 is accepted by $N_1$. The possible paths taken by the automaton on input 11 are:

- $q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_1$
- $q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2$
- $q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{\epsilon} q_3$
- $q_1 \xrightarrow{0} q_2 \xrightarrow{\epsilon} q_3 \xrightarrow{1} q_4$

The last path here is an accept path, hence $11 \in L(N_1)$.

Let us check similarly for 10:
• \( q_1 \xrightarrow{1} q_1 \xrightarrow{0} q_1 \)
• \( q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \)

None of these are accept paths, hence \( 10 \not\in L(N_1) \).

Observing carefully, we see that any path from \( q_1 \) to \( q_4 \) must consume a substring 11 or 101. And since \( q_1 \) and \( q_4 \) contain self-loops with both 0 and 1, all strings are allowed on either side. Hence, \( L(N_1) \) is the set of all strings containing either 11 or 101 as a substring.

**Example 2** Let \( L \) be a language consisting of all strings over \( \{0, 1\} \) such that the third-from-last digit is 1.

To do this, we can use this idea: first we accept any string, then we accept a 1, and then 2 more input characters which could be 0/1.

This gives the following NFA:

```
0,1
start ---\( q_1 \) \( 1 \) \( q_2 \) \( 0, 1 \) \( q_3 \) \( 0,1 \) \( q_4 \)
```

Let us do a sanity check with a couple of strings in the language, to check if each of them has **at least one** accept path in \( N_2 \).

• String: 01101. Accept path: \( q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \xrightarrow{1} q_4 \)
• String: 1111. Accept path: \( q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \xrightarrow{0} q_4 \xrightarrow{1} q_4 \)

Now consider the string 1001. It’s possible paths are:

• \( q_1 \xrightarrow{1} q_1 \xrightarrow{0} q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \)
• \( q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \xrightarrow{0} q_4 \xrightarrow{1} q_4 \)

None of these are accepting paths, so our NFA checks out in this case.

**Question:** Can we construct a DFA for the same?