Last lecture we proved the theorem:

The class of regular languages is closed under the union operation.

Proof:
- By construction
- Involved simulation
- Transition function involved PAIRS
Our next goal is to prove this theorem:

The class of regular languages is closed under the concatenation operation.

Iow:

If $A_1$ & $A_2$ are regular languages, then so is $A_1 \circ A_2$. 
Proof Idea (attempt 1)

* Follow the idea along the lines of the previous theorem (unim).
* Start with finite automata $M_1$ & $M_2$ recognizing languages $A_1$ & $A_2$.
* Construct an automaton $M$ that would accept input $w$ if:
  - $M_1$ accepts first piece of $w$ & $M_2$ accepts second piece of $w$.

**Problem:**

$M$ does not know where to break the input.

$w = s_1 s_2 s_3 \ldots \ s_n$
To solve this problem, we introduce

NON DETERMINISM
(Side box) - Your typical code

```javascript
function my_function (input) {
  Statement 1;
  Statement 2;
  ...
  Statement n;
  return my_result
}
```
Non-determinism is a generalization of determinism. So every DFA is automatically NFA.
Additional features in NFA (compared to DFA)

* I illustrate with this example N1

** Differences 1**

**DFA**

* Every state has exactly one exiting arrow for each symbol.

**NFA**

* Not true: case has 0.

  * Q1 has one exiting arrow for 0, but it has two for 1.
  * Q2 has one arrow for 0, but it has none for 1.
In general, in an NFA a state may have zero, one or many exiting arrows for each alphabet symbol.
Difference 2

<table>
<thead>
<tr>
<th>DFA</th>
<th>NFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>- labels on transition arrows are symbols from the alphabet.</td>
<td>- $N_1$ violates this. It has an arrow labelled with $\varepsilon$.</td>
</tr>
</tbody>
</table>

In general,

An NFA may have arrows labelled with members of the alphabet or $\varepsilon$.

Zero, one, or many arrows may exit from each state with the label $\varepsilon$. 
How does an NFA compute?

* Suppose we are running an NFA on an input string and we come to a state with multiple ways to proceed.
  For example, say we are in state $q_1$ in the NFA $M$ and the next input symbol is a $a$.

* After reading that symbol, the machine splits into multiple copies of itself and follows all possibilities in parallel.
Each copy of the machine takes one of the possible ways to proceed and continue as before.

If there are subsequent choices, the machine splits again.

If the next input symbol doesn't appear on any of the arrows exiting the state occupied by a copy of the machine, that copy of the machine dies.
Finally, if any one of those copies of the machine is an accept state at the end of the input, the NFA accepts the input string.
Views of Non-determinism

1. Kind of Parallel Computation

When the NFA splits to follow several choices, that corresponds to a process "forking" into several children, each processing separately.

If at least one of these processes accepts, then the entire computation accepts.
2) Tree of possibilities

1) Root corresponds to start of computation.

2) Each branching point in the tree corresponds to a point in the computation at which the machine has multiple choices.

3) Machine accepts if at least one of the branches ends in an accept state.
Convince yourself that $N_1$ accepts all strings that contain either 101 or 11 as a substring.
Example (N₁ revisited)

What would N₁ do on the input 010110?
Non-determinism is useful in several respects.

- As we will see, every NFA can be converted into an equivalent DFA, and constructing NFAs is sometimes easier than directly constructing DFAs.

- An NFA may be much smaller than its deterministic counterpart, or its functioning may be easier to understand.

- Non-determinism in DFA is also a good introduction to non-determinism in more powerful computational models.
Example

Let $A$ be the language consisting of all strings over $\{0,1\}$ containing a 1 in the third position from the end (e.g. 000100 is in $A$ but 0011 is not).

The following four-state NFA $N_2$ recognizes $A$.

![Diagram of NFA]

One good way to view the computation of this NFA is guessing the third symbol from the end is a 1. Use $q_2$ for the guess, and $q_3$ for the confirmation.
Every NFA can be converted into an equivalent DFA, but sometimes that DFA may have many more states.
The smallest DFA for A (above) contains 8 explicit states, and it is a lot more complex to understand.
Another example

The following NFA $N_3$ has an input alphabet $\{0\}$ consisting of a single symbol ( unary alphabet ).

This machine demonstrates the convenience of having $\varepsilon$ moves.

What language does this machine recognize?

$\{0^n \mid n \text{ is a multiple of } 2 \text{ or } 3\}$
A Third Example

Exercise: Convince yourself that it accepts the strings

\[\varepsilon, a, baba, baa\]
Formal Defn of NFA

* Similar to DFA, but differs in one essential way: the type of the transition function.

DFA

\[ \text{transition function} \quad (\text{state, input symbol}) \rightarrow (\text{next state}) \]

NFA

\[ (\text{state, input symbol}) \quad \vee \quad (\text{state, the empty string}) \rightarrow \text{set of possible next states} \]
To write a formal definition, we set up some additional notations.

* For any set \( \mathcal{Q} \) we write \( P(\mathcal{Q}) \) to be the collection of all subsets of \( \mathcal{Q} \).

* For any alphabet \( \Sigma \) we write \( \Sigma_\varepsilon \) to be \( \Sigma \cup \{\varepsilon\} \).

* Now we can write the formal description of the type of transition function in an NFA as

\[ s: \mathcal{Q} \times \Sigma_\varepsilon \rightarrow P(\mathcal{Q}) \]
Definition of NFA

An NFA is 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. \( \delta : Q \times \Sigma_e \rightarrow P(Q) \) is the transition function,
4. \( q_0 \in Q \) is the start state, and
5. \( F \subseteq Q \) is the set of accept states.
Formal definition of Computation for an NFA

(Similar to that of DFA.)

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and $w$ a string over the alphabet $\Sigma$.

We say $N$ accepts $w$ if we can write $w$ as $w = y_1 y_2 \ldots y_m$, where each $y_i$ is a member of $\Sigma^*$ and a sequence of $r_0, r_1, \ldots r_{m-1}$ in $Q$ with three conditions

1. $r_0 = q_0$
2. $r_i \in \delta(r_{i-1}, y_i)$, for $i = 0, \ldots, m-1$
3. $r_m \in F$