Previously:
- Introduced finite automata as a model of computation

Today:
- Quiz
- Discuss the regular languages
- Introduce nondeterminism
- Assignment 1

### Regular languages

**DEFINITION.** A language \( L \) is called a *regular language* if there exists a DFA \( D \) such that \( L(D) = L \), i.e., there is a DFA that recognizes it.
Regular operations

Union (∪):

A ∪ B = {w | w ∈ A or w ∈ B}

Concatenation (∘):

A ∘ B = AB = {xy | x ∈ A, y ∈ B}

Kleene star (*):

A* = {w | w = x1x2...xk, k ≥ 0, xi ∈ A}

Example of regular operations

If A = {a, b} and B = {b, c}, we get

A ∘ B = {ab, ac, bb, bc}

Note for *, we stick together symbols in any way we want to get longer string

A = {a, b, c}
A* = {ε, a, b, c, aa, bb, ab, ba, ac, bc, aba, abb, ...

We get an infinite language unless A ⊆ {ε}

Note ε ∈ A*, even if ε ∉ A

Closure properties of regular languages

The collection of regular languages is closed under regular operations.

If we take 2 regular languages – or 1 regular language, for Kleene star – and apply a regular operation, we get another regular language.

“Closed” means “you can’t get out” by using the operation.

Starting from a simple set of regular languages and applying these three operations, we can get to all regular languages.

Proof of closure under ∪

Show that if A and B are regular, then so is A ∪ B.

Proof by construction:

Given the automata

M1 = (Q1, Σ, δ1, q1, F1) recognizing A
M2 = (Q2, Σ, δ2, q2, F2) recognizing B

construct M = (Q, Σ, δ, q0, F) recognizing A ∪ B.

For simplicity, let the alphabets be the same.
You might think: Run the string through $M_1$, see whether $M_1$ accepts it, then run the string through $M_2$ and see whether $M_2$ accepts it.

But you can't try something on the whole input string, and try another thing on the whole input string; you get only 1 pass!

**Approach:** Simulate $M_1$ and $M_2$ simultaneously as we read each input symbol.

Imagine putting two fingers on the diagrams of the automata for $M_1$ and $M_2$, and moving them around according to the input.

At the end, if either finger is on an accept state, then we accept.

Implement this strategy in $M$.

**Formalization**

Keep track of a state in $M_1$ and a state in $M_2$ as a single state in $M$.

Each state in $M$ corresponds to a pair of states, one in $M_1$ and one in $M_2$.

Let $Q = Q_1 \times Q_2 = \{(q, r) \mid q \in Q_1, r \in Q_2\}$

How to define $\delta$?

**Define $\delta$**

When a new symbol comes in, go to wherever $q$ goes and wherever $r$ goes, individually

$\delta((q, r), a) = (\delta_1(q, a), \delta_2(r, a))$

Start state is $q_0 = (q_1, q_2)$

Accept set is $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Note $F_1 \times F_2$ would give the intersection, which isn't what we want.

It's clear by induction that the $k^{th}$ state of $M$ is just the $k^{th}$ state of $M_1$ and $k^{th}$ state of $M_2$. 
Problem

Prove that the collection of regular languages is closed under concatenation.

- But... where does the first string end and the second begin?
- As with union, the solution involves “keeping track of multiple possibilities”
- Next we’ll consider a type of finite automaton that can keep track of multiple possibilities, much simplifying writing these proofs.

Nondeterministic finite automata

All of the FAs we have seen so far are deterministic finite automata (DFAs)

- Exactly one possible move from each state to another for a given input symbol.

Nondeterministic finite automata

- Allow a finite automaton to have a choice of zero or more next states for each state–input pair.

Note: two “1” arrows from $q_0$ can take either.
The present state does not determine the next state; there are multiple possible futures!
NFAs are structurally similar to DFAs, but they represent a fundamental shift in how we’ll think about computation.

Example

\[
\begin{align*}
\text{start} & \rightarrow q_0 \\
1 & \rightarrow q_1 \\
1 & \rightarrow q_2 \\
\emptyset & \rightarrow q_3
\end{align*}
\]

\[
010110
\]

Begin at start state \(q_0\).
Read \(\emptyset\): Follow the loop back to \(q_0\).

Example

\[
\begin{align*}
\text{start} & \rightarrow q_0 \\
0, 1 & \rightarrow q_1 \\
1 & \rightarrow q_2 \\
\emptyset & \rightarrow q_3
\end{align*}
\]

\[
010110
\]

Read 1: There are two arrows labeled 1 starting at \(q_0\).
Split into two paths to represent the two places machine could be: \(q_0\) and \(q_1\).
Read 0: Now each path proceeds independently; they represent different threads of computation.

The path at q₀ goes back to q₀.

There's no place for the path at q₁ to go (no arrow with 0), so remove that path.

Read 1: Branch into q₀, q₁.

Read 1: Follow 1 arrows from q₀ and q₁ to get to q₀, q₁, q₂

Read 0: Follow 0 arrows from q₀, q₁, q₂ to get to q₀, q₃
The NFA accepts this string because at least one path ended up at an accepting state (q₃), i.e., 010110 ∈ L(B).

Each path represents a different thread of the computation.

By contrast, 01011 is not in L(B) because paths end at q₀, q₁, q₂; all possibilities are reject states.

Perfect guessing

We can view nondeterministic machines as having magic superpowers that enable them to guess the correct choice of moves to make.

If there is at least one choice leading to an accepting state for the input, the machine will guess it.

If there are no choices, the machine guesses any one of the wrong answers.

There's no physical analog for this style of computation.
Tree computation

We can view nondeterminism as a tree, where at each decision point, the automaton clones itself for each possible decision.

The series of choices forms a directed, rooted tree.
At the end, if any active accepting states remain, we accept.

Massive parallelism

An NFA can be thought of as a DFA that can be in many states at once.

Each symbol read causes a transition on every active state into each potential state that could be visited.

Nondeterministic machines can be thought of as machines that can try any number of options in parallel.
We’ll return to these intuitions to consider nondeterminism for different models through the semester.

Sometimes nondeterminism gives additional power; sometimes it doesn’t.

Problem

Design an NFA to accept strings over alphabet \{1, 2, 3\} such that the last symbol appears previously, without any intervening higher symbol, e.g.,

...11
...21112
...312123

Trick: Use the start state to mean “I guess I haven’t seen the symbol that matches the ending symbol yet”.

Three other states represent a guess that the matching symbol has been seen, and remembers that symbol.
Formal definition of a nondeterministic finite automaton (NFA)

\[ N = (Q, \Sigma, \delta, q_0, F) \]

like a DFA, but:

- \( \delta(q, a) \) is a set of states, rather than a single state

Extension to \( \hat{\delta} \):

- **Basis**: \( \hat{\delta}(q, \epsilon) = \{q\} \)
- **Induction**: Let
  \[
  \hat{\delta}(q, w) = \{p_1, p_2, \ldots, p_k\} \\
  \delta(p_i, a) = S_i \quad \text{for } i = 1, 2, \ldots, k \\
  \text{Then } \hat{\delta}(q, wa) = S_1 \cup S_2 \cup \ldots \cup S_k
  \]

Language of an NFA: An NFA accepts \( w \) if any path from the start state to an accepting state is labeled \( w \). Formally:

\[ L(N) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\} \]