15-251
Great Theoretical Ideas in Computer Science
Lecture 3:
Deterministic Finite Automata (DFA)

September 8th, 2015
What is computation?

What is an algorithm?

How can we mathematically define them?
Let’s assume two things about our world

No universal machines exist.

We only have machines to solve decision problems.
What is computation?

What is an algorithm?

How can we mathematically define them?

Today:

How do we represent information/data?

What is a computational problem?

Introducing deterministic finite automata (DFA)
Examples of computational problems

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>0</td>
</tr>
<tr>
<td>0, 1</td>
<td>1</td>
</tr>
<tr>
<td>1, 1</td>
<td>2</td>
</tr>
<tr>
<td>2, 2</td>
<td>4</td>
</tr>
<tr>
<td>2, 3</td>
<td>5</td>
</tr>
<tr>
<td>10, 1</td>
<td>11</td>
</tr>
<tr>
<td>100, 99</td>
<td>199</td>
</tr>
</tbody>
</table>

...
Examples of computational problems

input data → Primality → output data

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>251</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Examples of computational problems

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Yes</td>
</tr>
<tr>
<td>10101</td>
<td>Yes</td>
</tr>
<tr>
<td>selfless</td>
<td>No</td>
</tr>
<tr>
<td>dammitimmad</td>
<td>Yes</td>
</tr>
<tr>
<td>parahaziramarizaraharap</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Examples of computational problems

Instance
[vanilla, mind, Ariel, yogurt, doesn’t]

Solution
[Ariel, doesn’t, mind, vanilla, yogurt]
Representing information

**Familiar idea:**

Information in a computer is represented with 0s and 1s.

Can encode/represent any kind of data (numbers, text, pairs of numbers, graphs, images, etc...) with a finite length binary string.
Representing information

\[ \Sigma = \{0, 1\} \]

\( \Sigma^* = \{ \text{the set of all finite length strings over } \Sigma \} \)

\[ \Sigma^* = \{ \varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \ldots \} \]

A subset \( L \subseteq \Sigma^* \) is called a language.
Representing information

\[ \Sigma = \{a, b\} \]
\[ \Sigma = \{a, b, c\} \]
\[ \Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \]
\[ \Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\} \]

Can use whichever is convenient.
What is a computational problem?

Let \( \Sigma = \{0, 1\} \).

The palindrome computational problem is:

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \epsilon )</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>000</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
</tbody>
</table>

...
What is a computational problem?

Let $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \#\}$.

The **multiplication** computational problem is:

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0#0</td>
<td>0</td>
</tr>
<tr>
<td>0#1</td>
<td>0</td>
</tr>
<tr>
<td>1#0</td>
<td>0</td>
</tr>
<tr>
<td>1#1</td>
<td>1</td>
</tr>
<tr>
<td>10#2</td>
<td>20</td>
</tr>
<tr>
<td>11#3</td>
<td>33</td>
</tr>
<tr>
<td>12345679#9</td>
<td>111111111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
What is a computational problem?

**Definition:** A *computational problem* is a function

\[ f : \Sigma^* \rightarrow \Sigma^*. \]

**Definition:** A *decision problem* is a function

\[ f : \Sigma^* \rightarrow \{0, 1\}. \]

No, Yes
False, True
Reject, Accept
What is a computational problem?

Important

There is a one-to-one correspondence between decision problems and languages.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>000</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

$L \subseteq \Sigma^*$

$L = \{\epsilon, 0, 1, 00, 11, 000, \ldots\}$
What is computation?

What is an algorithm?

How can we mathematically define them?

Today:

How do we represent information/data?

What is a computational problem?

Introducing deterministic finite automata (DFA)
What is computation?

What is an algorithm?

How can we mathematically define them?

Today:

How do we represent information/data?
What is a computational problem?

Introducing deterministic finite automata (DFA)
Introducing deterministic finite automata (DFA)

- restricted model of computation
- very limited memory
  - reads input from left to right, and accepts or rejects. (one pass through the input)
\[ \Sigma = \{0, 1\} \]
State diagram of a DFA

\[ \Sigma = \{0, 1\} \]
State diagram of a DFA

\[ \Sigma = \{0, 1\} \]
\[ \Sigma = \{0, 1\} \]

Input: 1010
\[ \Sigma = \{0, 1\} \]

**Input:** 1010

---

Simulation of a DFA

\[ \Sigma = \{0, 1\} \]

**Input:** 1010
Simulation of a DFA

\[ \Sigma = \{0, 1\} \]

Input: 1010
Simulation of a DFA

$\Sigma = \{0, 1\}$

Input: 1010
$\Sigma = \{0, 1\}$

**Input:** 1010
Simulation of a DFA

$\Sigma = \{0, 1\}$

Input: 1010
Simulation of a DFA

\[ \Sigma = \{0, 1\} \]

Input: 1010
Simulation of a DFA

$\Sigma = \{0, 1\}$

Input: 1010
Simulation of a DFA

\[ \Sigma = \{0, 1\} \]

Input: 1010

Decision: Reject

\[
\begin{array}{c}
\text{q}_0 \\
\text{q}_1 \\
\text{q}_2 \\
\text{F}
\end{array}
\]
\( \Sigma = \{0, 1\} \)

Input: 01111
\[ \Sigma = \{0, 1\} \]

Input: 01111
\( \Sigma = \{0, 1\} \)

Input: 01111
$\Sigma = \{0, 1\}$

Input: 01111
\[ \Sigma = \{0, 1\} \]

**Input:** 01111
\[ \Sigma = \{0, 1\} \]

Input: 01111
Simulation of a DFA

\[ \Sigma = \{0, 1\} \]

Input: 01111
Input: 01111
Decision: Accept
Anatomy of a DFA

transition rule: labeled arrows
def foo(input):
    i = 0;

    STATE 0:
    if (i == input.length): return False;
    letter = input[i];
    i++;
    switch(letter):
        case '0': go to STATE 0;
        case '1': go to STATE 1;

    STATE 1:
    if (i == input.length): return True;
    letter = input[i];
    i++;
    switch(letter):
        case '0': go to STATE 2;
        case '1': go to STATE 2;
    
    ...
def foo(input):
    i = 0;
    STATE 0:
    if (i == input.length): return False;
    letter = input[i];
    i++;
    switch(letter):
        case '0': go to STATE 0;
        case '1': go to STATE 1;

    STATE 1:
    if (i == input.length): return True;
    letter = input[i];
    i++;
    switch(letter):
        case '0': go to STATE 2;
        case '1': go to STATE 2;

    ...
DFA as a programming language

```python
def foo(input):
    i = 0;
    STATE 0:
        if (i == input.length): return False;
        letter = input[i];
        i++;
        switch(letter):
            case '0': go to STATE 0;
            case '1': go to STATE 1;

    STATE 1:
        if (i == input.length): return True;
        letter = input[i];
        i++;
        switch(letter):
            case '0': go to STATE 2;
            case '1': go to STATE 2;
    
    # Continue state transitions...
```

input = [0, 1, 1, 1, 1]

Read current letter.
def foo(input):
    i = 0;

    # STATE 0:
    if (i == input.length): return False;
    letter = input[i];
    i++;
    switch(letter):
        case '0': go to STATE 0;
        case '1': go to STATE 1;

    # STATE 1:
    if (i == input.length): return True;
    letter = input[i];
    i++;
    switch(letter):
        case '0': go to STATE 2;
        case '1': go to STATE 2;

    # Continue...

Depending on the letter change the state.
def foo(input):
    i = 0;

    STATE 0:
    if (i == input.length): return False;
    letter = input[i];
    i++;  
    switch(letter):
        case '0': go to STATE 0;
        case '1': go to STATE 1;

    STATE 1:
    if (i == input.length): return True;
    letter = input[i];
    i++;  
    switch(letter):
        case '0': go to STATE 2;
        case '1': go to STATE 2;

    ...
Let $M$ be a DFA.

We let $L(M)$ denote the set of strings that $M$ accepts.

So, $L(M) = \{x \in \Sigma^* : M(x) \text{ accepts}\} \subseteq \Sigma^*$

If $L = L(M)$, we say that $M$ decides $L$. $M$ computes $L$.
$M$ recognizes $L$.
$M$ accepts $L$. 
DFA Examples

$L(M) = \text{ all binary strings with an even number of 1's}$

$= \{ x \in \{0, 1\}^* : x \text{ has an even number of 1's} \}$
$L(M) = \text{all binary strings with even length}
= \{ x \in \{0, 1\}^* : |x| \text{ is even} \}$
$L(M) = \{ x \in \{0, 1\}^* : x \text{ ends with a 0} \} \cup \{\epsilon\}$
DFA Examples

\[ M \]

\[ L(M) = \{a, b, cb, cc\} \]
Draw a DFA that decides

\[ L = \{ x \in \{0, 1\}^* : x \text{ starts and ends with same bit.} \} \]

**Hint:** How do you decide all strings that end with a 0 ?

How do you decide all strings that end with a 1 ?
The set of all words that contain at least three 0’s
The set of all words that contain at least two 0’s
The set of all words that contain 000 as a substring
The set of all words that contain 000 as a substring
The set of all words that contain 00 as a substring
The set of all words ending in 000
The set of all words ending in 00
None of the above
Beats me
DFA construction practice

\[ L = \{110, 101\} \]

\[ L = \{0, 1\}^* \backslash \{110, 101\} \]

\[ L = \{x \in \{0, 1\}^* : x \text{ starts and ends with same bit.}\} \]

\[ L = \{x \in \{0, 1\}^* : |x| \text{ is divisible by 2 or 3.}\} \]

\[ L = \{\epsilon, 110, 110110, 110110110, \ldots\} \]

\[ L = \{x \in \{0, 1\}^* : x \text{ contains the substring 110.}\} \]

\[ L = \{x \in \{0, 1\}^* : 10 \text{ and 01 occur equally often in } x.\} \]
A deterministic finite automaton (DFA) $M$ is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

where

- $Q$ is a finite set (which we call the set of states);
- $\Sigma$ is a finite set (which we call the alphabet);
- $\delta$ is a function of the form $\delta : Q \times \Sigma \rightarrow Q$ (which we call the transition function);
- $q_0 \in Q$ is an element of $Q$ (which we call the start state);
- $F \subseteq Q$ is a subset of $Q$ (which we call the set of accepting states).
Formal definition: DFA

A deterministic finite automaton (DFA) $M$ is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- $\delta : Q \times \Sigma \rightarrow Q$

$q_0$ is the start state

$F = \{q_1, q_2\}$
Formal definition: DFA accepting a string

Let $w = w_1w_2 \cdots w_n$ be a string over an alphabet $\Sigma$. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA.

We say that $M$ accepts the string $w$ if there exists a sequence of states $r_0, r_1, \ldots, r_n \in Q$ such that

- $r_0 = q_0$;
- $\delta(r_{i-1}, w_i) = r_i$ for each $i \in \{1, 2, \ldots, n\}$;
- $r_n \in F$.

Otherwise we say $M$ rejects the string $w$. 
Definition: A language $L$ is called regular if $L = L(M)$ for some DFA $M$. 
Regular languages

All languages
\( \mathcal{P}(\Sigma^*) \)

Regular languages

\{110, 101\}
\{0, 1\}^* \setminus \{110, 101\}
\{x \in \{0, 1\}^* : x \text{ starts and ends with same bit.}\}
\{x \in \{0, 1\}^* : |x| \text{ is divisible by 2 or 3.}\}
\{\epsilon, 110, 110110, 110110110, \ldots\}
\{x \in \{0, 1\}^* : x \text{ contains the substring 110.}\}
\{x \in \{0, 1\}^* : 10 \text{ and 01 occur equally often in } x.\}\
Questions:

1. Are all languages regular?  
   (Are all decision problems computable by a DFA?)

2. Are there other ways to tell if a language is regular?
A non-regular language

**Theorem:**
The language \( L = \{0^n1^n : n \in \mathbb{N}\} \) is **not** regular.

**Note on notation:**
- For \( a \in \Sigma \), \( a^n \) denotes the string \( aa \cdots a \) \( n \) times.
- \( a^0 = \epsilon \)
- For \( u, v \in \Sigma^* \), \( uv \) denotes \( u \) concatenated with \( v \).

So \( L = \{\epsilon, 01, 0011, 000111, 00001111, \ldots\} \).
Theorem:
The language $L = \{0^n1^n : n \in \mathbb{N}\}$ is not regular.

Intuition:
Seems like the DFA would need to remember how many 0's it sees.

But it has a constant number of states.
And no other way of remembering things.

Careful though:
$L = \{x \in \{0, 1\}^* : 10 \text{ and } 01 \text{ occur equally often in } x\}$ is regular!
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

imagine some arbitrary transitions
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

Imagine some arbitrary transitions
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

Imagine some arbitrary transitions
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides \( L = \{0^n1^n : n \in \mathbb{N}\} \).

Input: 0000000011111111
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

Imagine some arbitrary transitions
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides \( L = \{0^n1^n : n \in \mathbb{N}\} \).

Input: \text{0000000011111111}

imagine some arbitrary transitions
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n1^n : n \in \mathbb{N}\}$.

Input: 0000000001111111

imagine some arbitrary transitions
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides \( L = \{0^n1^n : n \in \mathbb{N}\} \).

Input: 0000000011111111

Imagine some arbitrary transitions
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n1^n : n \in \mathbb{N}\}$.

**Input:** 0000000011111111

- $q_0$
- $q_1$
- $q_2$
- $q_3$
- $q_4$
- $q_5$

Imagine some arbitrary transitions.
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

imagine some arbitrary transitions
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

imagine some arbitrary transitions
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

Imagine some arbitrary transitions
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides \( L = \{0^n1^n : n \in \mathbb{N}\} \).

Input: 00000000|1|1|1|1|1|1|1|1|1

imagine some arbitrary transitions
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

After 00 and 000000 we ended up in the same state $q_3$.

0011 and 00000011 end up in the same state.

imagine some arbitrary transitions

But
0011 → accept
00000011 → reject
A non-regular language

Warm-up:
Suppose a DFA with 6 states decides

\[ L = \{ 0^n 1^n : n \in \mathbb{N} \} \]

Input: \[ 0000000011111111 \]

Where will 0000000 go?

Imagine some arbitrary transitions

Pigeonhole Principle
Theorem: The language \( L = \{0^n1^n : n \in \mathbb{N}\} \) is not regular.

Proof: Suppose \( L \) is regular. So there is a DFA \( M \) that decides \( L \).
Let \( k \) denote the number of states of \( M \).
Let \( r_n \) denote the state \( M \) is in after reading \( 0^n \).
By PHP, there exists \( i, j \in \{0, 1, \ldots, k\}, i \neq j \), such that \( r_i = r_j \). So \( 0^i \) and \( 0^j \) end up in the same state.
For any string \( w \), \( 0^i w \) and \( 0^j w \) end up in the same state.
But for \( w = 1^i \), \( 0^i w \) should end up in an accepting state, and \( 0^j w \) should end up in a rejecting state.
This is the desired contradiction.
Proving a language is not regular

Usually the proof goes like:

1. Assume (to reach a contradiction) that the language is regular. So a DFA decides it.

2. Argue by PHP that there are two strings $x$ and $y$ that lead to the same state in the DFA.

3. Find a string $z$ such that $xz \in L$ but $yz \notin L$. 
All languages
\( \mathcal{P}(\Sigma^*) \)

Regular languages

- \{110, 101\}
- \{0, 1\}^* \setminus \{110, 101\}
- \{x \in \{0, 1\}^* : x \text{ starts and ends with same bit.}\}
- \{x \in \{0, 1\}^* : |x| \text{ is divisible by 2 or 3.}\}
- \{\epsilon, 110, 110110, 110110110, \ldots\}
- \{x \in \{0, 1\}^* : x \text{ contains the substring 110.}\}
- \{x \in \{0, 1\}^* : 10 \text{ and 01 occur equally often in } x.\}
  
  ...
Regular languages

All languages
$\mathcal{P}(\Sigma^*)$

Regular languages

$\{0^n1^n : n \in \mathbb{N}\}$

- $\{110, 101\}$
- $\{0, 1\}^* \setminus \{110, 101\}$
- $\{x \in \{0, 1\}^* : x \text{ starts and ends with same bit.}\}$
- $\{x \in \{0, 1\}^* : |x| \text{ is divisible by 2 or 3.}\}$
- $\{\epsilon, 110, 110110, 110110110, \ldots\}$
- $\{x \in \{0, 1\}^* : x \text{ contains the substring 110.}\}$
- $\{x \in \{0, 1\}^* : 10 \text{ and 01 occur equally often in } x.\}$
- $\ldots$
Questions:

1. Are all languages regular?  
   (Are all decision problems computable by a DFA?)

2. Are there other ways to tell if a language is regular?
Regular languages are closed under union

For $L_1, L_2 \subseteq \Sigma^*$,

$$L_1 \cup L_2 = \{ x \in \Sigma^* : x \in L_1 \text{ or } x \in L_2 \}$$

**Theorem:**

Let $\Sigma$ be some finite alphabet.

If $L_1, L_2 \subseteq \Sigma^*$ are regular, then so is $L_1 \cup L_2$.

**Proof:** Let $M = (Q, \Sigma, \delta, q_0, F)$ be the decider for $L_1$ and $M' = (Q', \Sigma, \delta', q'_0, F')$ be the decider for $L_2$.

We construct a DFA $M'' = (Q'', \Sigma, \delta'', q''_0, F'')$ that decides $L_1 \cup L_2$, as follows:

\[\vdots\]
Regular languages are closed under union

Example

\[ L_1 = \] strings with even number of 1’s

\[ L_2 = \] strings with length divisible by 3.

\[ M_1 \]

\[ q_{even} \]

\[ 0 \]

\[ 1 \]

\[ 1 \]

\[ 0 \]

\[ q_{odd} \]

\[ M_2 \]

\[ p_0 \]

\[ p_1 \]

\[ p_2 \]
Regular languages are closed under union

Input: 101001

\[ M_1 \]

\[ q_{\text{even}} \]

\[ q_{\text{odd}} \]

\[ M_2 \]

\[ p_0 \]

\[ p_1 \]

\[ p_2 \]
Regular languages are closed under union

Input: 101001

\[ M_2 \]

\[ p_0 \]

\[ 0, 1 \]

\[ 0, 1 \]

\[ p_1 \]

\[ 0, 1 \]

\[ p_2 \]

\[ M_1 \]

\[ q_{even} \]

\[ 0 \]

\[ 1 \]

\[ 1 \]

\[ 0 \]

\[ q_{odd} \]
Regular languages are closed under union

Input: 101001

M₁

q_{even}

0

1

1

0

q_{odd}

M₂

p₀

0, 1

0, 1

p₁

0, 1

p₂
Regular languages are closed under union

Input: 101001
Regular languages are closed under union

Input: $101001$

$M_1$

$q_{even}$

0 \rightarrow 1 \rightarrow 1

$q_{odd}$

0

$M_2$

$p_0$

0, 1

$p_1$

0, 1

$p_2$

0, 1
Regular languages are closed under union

Input: 101001
Regular languages are closed under union

Input: 101001

$M_1$
$q_{\text{even}}$

$M_2$
$p_0$

$q_{\text{odd}}$

$p_1$

$p_2$
Regular languages are closed under union

Input: 101001
Regular languages are closed under union

Input: $101001$

Accept
Main idea:
Construct a DFA that keeps track of both at once.
Regular languages are closed under union

Main idea:
Construct a DFA that keeps track of both at once.
Main idea:
Construct a DFA that keeps track of both at once.
Regular languages are closed under union

Main idea:
Construct a DFA that keeps track of both at once.
Regular languages are closed under union

Main idea:
Construct a DFA that keeps track of both at once.
Regular languages are closed under union

Main idea:
Construct a DFA that keeps track of both at once.
Regular languages are closed under union

Main idea:
Construct a DFA that keeps track of both at once.
Regular languages are closed under union

Main idea:
Construct a DFA that keeps track of both at once.
Regular languages are closed under union

Main idea:
Construct a DFA that keeps track of both at once.

```
q_{even \ p_0} \xrightarrow{0} q_{even \ p_1} \xrightarrow{1} q_{odd \ p_0} \xrightarrow{1} q_{odd \ p_1} \xrightarrow{0} q_{odd \ p_2}
```

Regular languages are closed under union

Main idea:
Construct a DFA that keeps track of both at once.
Main idea:
Construct a DFA that keeps track of both at once.
Regular languages are closed under union.

Input: 101001
Regular languages are closed under union

Input: 101001

The diagram shows a deterministic finite automaton (DFA) with states labeled as even and odd, and transitions labeled with 0s and 1s. The input string 101001 is shown moving through the states, starting from $q_{even} p_0$, through $q_{even} p_1$, and ending in $q_{odd} p_2$, indicating that the string is accepted by the DFA.
Regular languages are closed under union

Input: 101001

- **$q_{even \ p_0}$**
  - Transitions: $0 \rightarrow q_{even \ p_0}, 1 \rightarrow q_{odd \ p_0}$

- **$q_{even \ p_1}$**
  - Transitions: $0 \rightarrow q_{even \ p_1}, 1 \rightarrow q_{odd \ p_1}$

- **$q_{even \ p_2}$**
  - Transitions: $0 \rightarrow q_{even \ p_2}, 1 \rightarrow q_{odd \ p_2}$

- **$q_{odd \ p_0}$**
  - Transitions: $0 \rightarrow q_{odd \ p_0}, 1 \rightarrow q_{even \ p_0}$

- **$q_{odd \ p_1}$**
  - Transitions: $0 \rightarrow q_{odd \ p_1}, 1 \rightarrow q_{even \ p_1}$

- **$q_{odd \ p_2}$**
  - Transitions: $0 \rightarrow q_{odd \ p_2}, 1 \rightarrow q_{even \ p_2}$
Regular languages are closed under union

Input: 101001
Regular languages are closed under union

Input: 101001
Regular languages are closed under union

Input: 101001
Regular languages are closed under union

Input: 101001
Regular languages are closed under union

Input: 101001
Regular languages are closed under union

Input: 101001
Regular languages are closed under union

Input: 101001
Regular languages are closed under union

Input: 101001
Regular languages are closed under union.
Regular languages are closed under union

Input: 101001
Regular languages are closed under union

Input: 101001
Decision: Accept
Regular languages are closed under union

**Theorem:**
Let \( \Sigma \) be some finite alphabet. If \( L_1 \subseteq \Sigma^* \) and \( L_2 \subseteq \Sigma^* \) are regular, then so is \( L_1 \cup L_2 \).

**Proof:** Let \( M = (Q, \Sigma, \delta, q_0, F) \) be the decider for \( L_1 \) and \( M' = (Q', \Sigma, \delta', q'_0, F') \) be the decider for \( L_2 \). We construct a DFA \( M'' = (Q'', \Sigma, \delta'', q''_0, F'') \) that decides \( L_1 \cup L_2 \), as follows:

1. \( Q'' = Q \times Q' = \{(q, q') : q \in Q, q' \in Q'\} \)
2. \( \delta''((q, q'), a) = (\delta(q, a), \delta'(q', a)) \)
3. \( q''_0 = (q_0, q'_0) \)
4. \( F'' = \{(q, q') : q \in F \text{ or } q' \in F'\} \)
Regular languages are closed under union

Proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be the decider for $L_1$ and $M' = (Q', \Sigma, \delta', q'_0, F')$ be the decider for $L_2$. We construct a DFA $M'' = (Q'', \Sigma, \delta'', q''_0, F'')$ that decides $L_1 \cup L_2$, as follows:

- $Q'' = Q \times Q' = \{(q, q') : q \in Q, q' \in Q'\}$
- $\delta''((q, q'), a) = (\delta(q, a), \delta'(q', a))$
- $q''_0 = (q_0, q'_0)$
- $F'' = \{(q, q') : q \in F$ or $q' \in F'\}$

It remains to show that $L(M'') = L_1 \cup L_2$.

$L(M'') \subseteq L_1 \cup L_2 : \ldots$

$L_1 \cup L_2 \subseteq L(M'') : \ldots$
More “closure” properties

Closed under union:

\[ L_1, L_2 \text{ regular } \implies L_1 \cup L_2 \text{ regular.} \]

Closed under concatenation:

\[ L_1, L_2 \text{ regular } \implies L_1 \cdot L_2 \text{ regular.} \]

\[ L_1 \cdot L_2 = \{xy : x \in L_1, y \in L_2\} \]

Closed under star:

\[ L \text{ regular } \implies L^* \text{ regular.} \]

\[ L^* = \{x_1x_2 \cdots x_k : k \geq 0, \forall i \ x_i \in L\} \]
More “closure” properties

Fact:

Starting with $\emptyset$ and $\{a\}$ for each $a \in \Sigma$ can construct any regular language using union, concatenation, and star.

$$a(a \cup b)^* a \cup b(a \cup b)^* b \cup a \cup b$$

(regular expression)
Next Time