CSE 303: Introduction to the Theory of Computation
(Finite Automata)

Pramod Ganapathi
Department of Computer Science
State University of New York at Stony Brook

February 12, 2020
Deterministic Finite Automata (DFA)
Regular Languages
Regular Expressions
Nondeterministic Finite Automata (NFA)
Transformations
Non-Regular Languages
Deterministic Finite Automata (DFA)
Problem

- Design the logic behind an electric bulb.
## Electric bulb

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design the logic behind an electric bulb.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>Diagram.</strong></td>
</tr>
</tbody>
</table>
| • **Analysis.**  
  States = \{nolight, light\}, Input = \{off, on\}  
  **Finite Automaton.** |
### Problem

- Design the logic behind a multispeed fan.
Multispeed fan

Problem

- Design the logic behind a multispeed fan.

Solution

- Diagram.
- Finite Automaton.

- Analysis.
  States = \{0, 1, 2, 3\}
  Input = \{⟳, ⟲\}
### Problem

- Design the logic behind automatic doors in Walmart.
Automatic doors

Solution

- **Diagram.**

- **Analysis.**
  States = \{close, open\}, Input = \{left, right, neither\}

- **Finite Automaton.**
Basic features of finite automata

- A finite automaton is a simple computer with \textit{extremely limited memory}.
- A finite automaton has a \textit{finite set of states}.
- \textbf{Current state} of a finite automaton changes when it reads an input symbol.
- A finite automaton acts as a \textit{language acceptor} i.e., outputs “yes” or “no”.
Why should you care?

Deterministic Finite Automata (DFA) are everywhere.

- ATMs
- Ticket machines
- Vending machines
- Traffic signal systems
- Calculators
- Digital watches
- Automatic doors
- Elevators
- Washing machines
- Dishwashing machines
- Thermostats
- Train switches
- (CS) Compilers
- (CS) Search engines
- (CS) Regular expressions
Why should you care?

Probabilistic Finite Automata (PFA) are everywhere, too.

- Speech recognition
- Optical character recognition
- Thermodynamics
- Statistical mechanics
- Chemical reactions
- Information theory
- Queueing theory
- PageRank algorithm
- Statistics
- Reinforcement learning
- Price changes in finance
- Genetics
- Algorithmic music composition
- Bioinformatics
- Probabilistic forecasting
What is a decision problem?

**Definition**

- A **decision problem** is a computational problem with a ‘yes’ or ‘no’ answer.
- A computer that solves a decision problem is a **decider**.
  - Input to a decider: A string $w$
  - Output of a decider: Accept ($w$ is in the language) or Reject ($w$ is not in the language)
What is a decision problem?

- Language = English language = \{milk, food, sleep, \ldots\} $\rightarrow$ Accept
- Not in language = \{zffgb, cdcapqw, \ldots\} $\rightarrow$ Reject
What is a decision problem?

Some strings $\rightarrow$ Accept

Other strings $\rightarrow$ Reject
How does a DFA work?

Problem

- Does the DFA accept the string $bbab$?

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>q1</td>
</tr>
</tbody>
</table>
```

Solution

The DFA accepts the string $bbab$. The computation is:

1. Start in state $q_0$.
2. Read $b$, follow transition from $q_0$ to $q_1$.
3. Read $b$, follow transition from $q_1$ to $q_1$.
4. Read $a$, follow transition from $q_1$ to $q_2$.
5. Read $b$, follow transition from $q_2$ to $q_1$.
6. Accept because the DFA is in an accept state $q_1$ at the end of the input.
How does a DFA work?

**Problem**

- Does the DFA accept the string $bbab$?

![DFA Diagram]

**Solution**

The DFA accepts the string $bbab$. The computation is:

1. Start in state $q_0$
2. Read $b$, follow transition from $q_0$ to $q_1$.
3. Read $b$, follow transition from $q_1$ to $q_1$.
4. Read $a$, follow transition from $q_1$ to $q_2$.
5. Read $b$, follow transition from $q_2$ to $q_1$.
6. Accept because the DFA is in an accept state $q_1$ at the end of the input.
Problem

- Does the DFA accept the string $aaba$?

### Solution

The DFA rejects the string $aaba$. The computation is:

1. Start in state $q_0$
2. Read $a$, follow transition from $q_0$ to $q_0$.
3. Read $a$, follow transition from $q_0$ to $q_0$.
4. Read $b$, follow transition from $q_0$ to $q_1$.
5. Read $a$, follow transition from $q_1$ to $q_2$.
6. Reject because the DFA is in a reject state $q_2$ at the end of the input.
How does a DFA work?

Problem

- Does the DFA accept the string \textit{aaba}?

Solution

The DFA rejects the string \textit{aaba}. The computation is:

1. Start in state \( q_0 \)
2. Read \( a \), follow transition from \( q_0 \) to \( q_0 \).
3. Read \( a \), follow transition from \( q_0 \) to \( q_0 \).
4. Read \( b \), follow transition from \( q_0 \) to \( q_1 \).
5. Read \( a \), follow transition from \( q_1 \) to \( q_2 \).
6. Reject because the DFA is in a reject state \( q_2 \) at the end of the input.
How does a DFA work?

The DFA processes the input strings "bbab" and "aaba". For "bbab",
- Start at state $q_0$.
- After reading 'b', move to state $q_1$.
- After reading 'b', move to state $q_2$.
- The input sequence is accepted because the DFA ends in an accept state.

For "aaba",
- Start at state $q_0$.
- After reading 'a', move to state $q_1$.
- After reading 'a', move to state $q_2$.
- The input sequence is rejected because the DFA does not end in an accept state.
How does a DFA work?

Problem

- What language does the DFA accept?

Examples

- The DFA accepts the following strings:
  - $b$, $ab$, $bb$, $aabb$, $ababababab$, ...
  - $baa$, $abaa$, $ababaaaaaa$, ...

- The DFA rejects the following strings:
  - $a$, $ba$, $babaaa$, ...
How does a DFA work?

### Problem

- What language does the DFA accept?

![DFA Diagram]

### Examples

- The DFA accepts the following strings:
  - $b, ab, bb, aabbbb, ababababab, \ldots$ $\triangleright$ ends with $b$
  - $baa, abaa, ababaaaaaa, \ldots$ $\triangleright$ ends with $b$ followed by even $a$’s

- The DFA rejects the following strings:
  - $a, ba, babaaa, \ldots$

- What language does the DFA accept?
Construct DFA for $\Sigma = \{a\}$

**Problem**

- Construct a DFA that accepts all strings from the language $L = \{\epsilon, a, aa, aaa, aaaa, \ldots\}$
Construct DFA for $\Sigma = \{a\}$

**Problem**

- Construct a DFA that accepts all strings from the language $L = \{\epsilon, a, aa, aaa, aaaa, \ldots\}$

**Solution**

- Language $L$: $\Sigma^* = \{\epsilon, a, aa,aaa,aaaa,\ldots\}$
- Expression: $a^*$
- Deterministic Finite Automaton (DFA) $M$:

```
    a

start --→ q0
```
Construct DFA for $\Sigma = \{a\}$

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Construct a DFA that accepts all strings from the language $L = {}$</td>
</tr>
</tbody>
</table>

Language $L$: $\emptyset = \{\}$ — Empty language

Expression: $\emptyset$

DFA $M$:
- $q_0$: start
- $a$
Problem

- Construct a DFA that accepts all strings from the language $L = \{\}$. 

Solution

- Language $L$: $\phi = \{}$
- Expression: $\phi$  
- DFA $M$:

```
  a

  q0
```

- $\triangleright$ Empty language
Construct DFA for $\Sigma = \{a\}$

Problem

- Construct a DFA that accepts all strings from the language $L = \{a, aa, aaa, aaaa, \ldots\}$
Problem

• Construct a DFA that accepts all strings from the language
  \[ L = \{ a, aa, aaa, aaaaa, \ldots \} \]

Solution

• Language \( L: \Sigma^* - \{ \epsilon \} = \{ a, aa, aaa, aaaaa, \ldots \} \)
• Expression: \( a^+ \)
• DFA \( M: \)

\[ q_0 \xrightarrow{a} q_1 \]

\[ q_1 \xrightarrow{a} q_1 \]
Construct DFA for $\Sigma = \{a\}$

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Construct a DFA that accepts all strings from the language $L = {\epsilon}$</td>
</tr>
</tbody>
</table>
Construct DFA for $\Sigma = \{a\}$

**Problem**
- Construct a DFA that accepts all strings from the language $L = \{\epsilon\}$

**Solution**
- Language $L: = \{\epsilon\}$
- Expression: $\epsilon$
- DFA $M$:

![DFA Diagram]

- Start state: $q_0$
- Final state: $q_1$
- Transition on $a$: $q_0 \rightarrow q_1$
Construct DFA for $\sum = \{a\}$

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Construct a DFA that accepts all strings from the language $L = {aaa}$</td>
</tr>
</tbody>
</table>
Problem

- Construct a DFA that accepts all strings from the language $L = \{aaa\}$

Solution

- Language $L$: $\{aaa\}$
- Expression: $aaa$
- DFA $M$:
Construct DFA for $\Sigma = \{a\}$

Problem

- Construct a DFA that accepts all strings from the language $L = \{\text{strings with even size}\}$
Construct DFA for $\Sigma = \{a\}$

Problem
- Construct a DFA that accepts all strings from the language $L = \{\text{strings with even size}\}$

Solution
- Language $L$: $\{\epsilon, aa, aaaaa, aaaaaaa, \ldots\}$
- Expression: $(aa)^*$
- DFA $M$:

```
  start ----> q0 ----> q1
             \   / a
              |    |
              \   / a
      q0 ----> q1
```
Problem

- Construct a DFA that accepts all strings from the language $L = \{\text{strings with odd size}\}$
### Problem

- Construct a DFA that accepts all strings from the language $L = \{ \text{strings with odd size} \}$

### Solution

- Language $L$: $\{a, aaa, aaaaaa, \ldots\}$
- Expression: $a(aa)^*$
- DFA $M$:

```
    q0 ----> a ----> q1
     |          |          |
     |          v          |
     |                a ----> q0
    start
```

Diagram showing:
- Initial state $q_0$ connected with an arrow labeled $a$ to $q_1$, indicating a transition on input $a$.
- Transition from $q_1$ back to $q_0$ on input $a$. This represents the DFA accepting strings of odd length by looping back to the start state after an odd number of $a$s, ensuring the final state is reached only after an odd number of inputs.
Problem

- Construct a DFA that accepts all strings from the language $L = \{\text{strings of size divisible by 3}\}$
Construct DFA for $\Sigma = \{a\}$

**Problem**

- Construct a DFA that accepts all strings from the language $L = \{\text{strings of size divisible by 3}\}$

**Solution**

- Language $L: \{\epsilon, aaa, aaaaaa, aaaaaaaaaa, \ldots\}$
- Expression: $(aaa)^*$
- DFA $M$:

![DFA Diagram]

- Start state: $q_0$
- Final states: $q_1$, $q_2$
- Transitions:
  - $q_0 \xrightarrow{a} q_1$
  - $q_1 \xrightarrow{a} q_2$
  - $q_2 \xrightarrow{a} q_0$
Problem

- Construct a DFA that accepts all strings from the language $L = \{\text{strings of size not divisible by 3}\}$
Construct DFA for $\Sigma = \{a\}$

**Problem**
- Construct a DFA that accepts all strings from the language $L = \{\text{strings of size not divisible by 3}\}$

**Solution**
- Language $L$: $\{a, aa, aaaa, aaaaaa, \ldots\}$
- Expression: $(a \cup aa)(aaa)^*$
- DFA $M$:

```
<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$a$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$a$</td>
<td>$q_2$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$q_0$</td>
</tr>
</tbody>
</table>
```

Diagram:
- Start at $q_0$
- On $a$: $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_0$
<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Construct a DFA that accepts all strings from the language $L = {\text{strings of size divisible by 6}}$</td>
</tr>
</tbody>
</table>
Construct DFA for $\Sigma = \{a\}$

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Construct a DFA that accepts all strings from the language $L = {\text{strings of size divisible by 6}}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Language $L$: ${\epsilon, aaaaaa, aaaaaaaaaaa, \ldots}$</td>
</tr>
<tr>
<td>• Expression: $(aaaaaa)^*$</td>
</tr>
<tr>
<td>• DFA $M$:</td>
</tr>
</tbody>
</table>

![DFA Diagram]

Can you think of another approach?
Construct DFA for \( \Sigma = \{a\} \)

**Problem**

- Construct a DFA that accepts all strings from the language \( L = \{ \text{strings of size divisible by 6} \} \)

**Solution**

- Language \( L: \{ \epsilon, aaaaaa, aaaaaaaaaaa, \ldots \} \)
- Expression: \((aaaaaa)^*\)
- DFA \( M: \)

```
start -----> q0 -----> a -----> q1 -----> a -----> q2 -----> a -----> q3 -----> a -----> q4 -----> a -----> q5
```

- Can you think of another approach?
Construct DFA for $\Sigma = \{a\}$

**Problem**
- Construct a DFA that accepts all strings from the language $L = \{\text{strings of size divisible by 6}\}$

**Solution**
- Let $n = \text{string size}$
- **Observation**
  \[ n \mod 6 = 0 \iff n \mod 2 = 0 \text{ and } n \mod 3 = 0 \]
- **Idea**
  Build DFA $M_1$ for $n \mod 2 = 0$.
  Build DFA $M_2$ for $n \mod 3 = 0$.
  Run $M_1$ and $M_2$ in parallel.
  Accept a string if both DFAs $M_1$ and $M_2$ accept the string.
  Reject a string if at least one of the DFAs $M_1$ and $M_2$ reject the string.
- **It is possible to build complicated DFAs from simpler DFAs**
Construct DFA for $\Sigma = \{a\}$

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Construct a DFA that accepts all strings from the language $L = {\text{strings with size } n \text{ where } n \mod 4 = 2}$</td>
</tr>
</tbody>
</table>
**Problem**

- Construct a DFA that accepts all strings from the language $L = \{\text{strings with size } n \text{ where } n \mod 4 = 2\}$

**Solution**

- Language $L$: \{aaa, aaaaaaa, aaaaaaaaaaa, ...\}
- Expression: $aa(aaaa)^*$
- DFA $M$:

```
  start ----> rem 0 ----> rem 1 ----> rem 2 ----> rem 3
```

- What about strings with size $n$ where $n \mod k = i$?
Construct DFA for $\Sigma = \{a\}$

More Problems

Construct a DFA that accepts all strings from the language $L = \{\text{strings with size } n\}$ such that

- $n^2 - 5n + 6 = 0$
- $n \in [4, 37]$
- $n$ is a perfect cube
- $n$ is a prime number
- $n$ satisfies a mathematical function $f(n)$
The specification of DFA consists of:
- A (finite) alphabet
- A (finite) set of states
- Which state is the start state?
- Which states are the final states?
- What is the transition from each state, on each input character?
What is a deterministic finite automaton (DFA)?

- Deterministic = Events can be determined precisely
- Finite = Finite and small amount of space used
- Automaton = Computing machine

A deterministic finite automaton (DFA) $M$ is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where,

1. $Q$: A finite set (set of states).
2. $\Sigma$: A finite set (alphabet).
3. $\delta$: $Q \times \Sigma \rightarrow Q$ is the transition function.
4. $q_0$: The start state (belongs to $Q$).
5. $F$: The set of accepting/final states, where $F \subseteq Q$. 


What is a deterministic finite automaton (DFA)?

- Deterministic = Events can be determined precisely
- Finite = Finite and small amount of space used
- Automaton = Computing machine

**Definition**

A deterministic finite automaton (DFA) $M$ is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where,

1. $Q$: A finite set (set of states).  ▶ Space (computer memory)
2. $\Sigma$: A finite set (alphabet).
3. $\delta: Q \times \Sigma \rightarrow Q$ is the transition function.
   ▶ Time (computation)
4. $q_0$: The start state (belongs to $Q$).
5. $F$: The set of accepting/final states, where $F \subseteq Q$. 

Acceptance and rejection of strings

A DFA accepts a string \( w = w_1w_2\ldots w_k \) iff there exists a sequence of states \( r_0, r_1, \ldots, r_k \) such that the current state starts from the start state and ends at a final state when all the symbols of \( w \) have been read.

A DFA rejects a string iff it does not accept it.
What is a regular language?

Definition

- We say that a DFA $M$ accepts a language $L$ if $L = \{w \mid M$ accepts $w\}$.
- A language is called a regular language if some DFA accepts or recognizes it.
Problem

• Construct a DFA that accepts all strings from the language $L = \{\text{strings with odd number of } b\text{'s}\}$
Construct DFA for $\Sigma = \{a, b\}$

**Problem**
- Construct a DFA that accepts all strings from the language $L = \{\text{strings with odd number of } b\text{'s}\}$

**Solution**

**States**
- $q_{\text{odd}}$: DFA is in this state if it has read odd $b$’s.
- $q_{\text{even}}$: DFA is in this state if it has read even $b$’s.
Construct DFA for $\Sigma = \{a, b\}$

Problem
- Construct a DFA that accepts all strings from the language $L = \{\text{strings with odd number of } b\text{'s}\}$

Solution
- Language $L$: $\{\text{strings with odd number of } b\text{'s}\}$
- Expression: $a^*b(a \cup ba^*b)^* \text{ or } a^*ba^*(ba^*ba^*)^*$
- DFA $M$:
Construct DFA for $\Sigma = \{a, b\}$

Problem

- Construct a DFA that accepts all strings from the language $L = \{\text{strings with odd number of } b\text{'s}\}$

Solution (continued)

- DFA $M$ is specified as
  - Set of states is $Q = \{q_{\text{even}}, q_{\text{odd}}\}$
  - Set of symbols is $\Sigma = \{a, b\}$
  - Start state is $q_{\text{even}}$
  - Set of accept states is $F = \{q_{\text{odd}}\}$

  Transition function $\delta$ is:

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_{\text{even}}$</td>
<td>$q_{\text{even}}$</td>
<td>$q_{\text{odd}}$</td>
</tr>
<tr>
<td>$q_{\text{odd}}$</td>
<td>$q_{\text{odd}}$</td>
<td>$q_{\text{even}}$</td>
</tr>
</tbody>
</table>
Problem

- Construct a DFA that accepts all strings from the language $L = \{\text{strings containing } bab\}$
Construct DFA for $\Sigma = \{a, b\}$

Problem

- Construct a DFA that accepts all strings from the language $L = \{\text{strings containing } bab\}$

Solution

States

- $q_b$: DFA is in this state if the last symbol read was $b$, but the substring $bab$ has not been read.
- $q_{ba}$: DFA is in this state if the last two symbols read were $ba$, but the substring $bab$ has not been read.
- $q_{bab}$: DFA is in this state if the substring $bab$ has been read in the input string.
- $q$: In all other cases, the DFA is in this state.
Problem

- Construct a DFA that accepts all strings from the language $L = \{\text{strings containing } bab\}$

Solution (continued)

- Language $L$: $\{\text{strings containing } bab\}$
- Expression: $(a^*b^+aa)^*bab(a \cup b)^*$
- DFA $M$:
Problem

- Construct a DFA that accepts all strings from the language $L = \{\text{strings containing} \ bab\}$

Solution (continued)

- DFA $M$ is specified as
  - Set of states is $Q = \{q, q_b, q_{ba}, q_{bab}\}$
  - Set of symbols is $\Sigma = \{a, b\}$
  - Start state is $q$
  - Set of accept states is $F = \{q_{bab}\}$

Transition function $\delta$ is:

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>$q$</td>
<td>$q_b$</td>
</tr>
<tr>
<td>$q_b$</td>
<td>$q_{ba}$</td>
<td>$q_b$</td>
</tr>
<tr>
<td>$q_{ba}$</td>
<td>$q$</td>
<td>$q_{bab}$</td>
</tr>
<tr>
<td>$q_{bab}$</td>
<td>$q_{bab}$</td>
<td>$q_{bab}$</td>
</tr>
</tbody>
</table>
## Closure properties of regular languages

**Properties**

Let $L_1$ and $L_2$ be regular languages. Then, the following languages are regular.

- **Complement.** $\overline{L_1} = \{ x \mid x \in \Sigma^* \text{ and } x \notin L_1 \}$.
- **Union.** $L_1 \cup L_2 = \{ x \mid x \in L_1 \text{ or } x \in L_2 \}$.
- **Intersection.** $L_1 \cap L_2 = \{ x \mid x \in L_1 \text{ and } x \in L_2 \}$.
- **Concatenation.** $L_1 \cdot L_2 = \{ xy \mid x \in L_1 \text{ and } y \in L_2 \}$.
- **Star.** $L_1^* = \{ x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in L_1 \}$. 
## Closure properties for languages

<table>
<thead>
<tr>
<th>Language</th>
<th>$L_1 \cup L_2$</th>
<th>$L_1 \cap L_2$</th>
<th>$L'$</th>
<th>$L_1 L_2$</th>
<th>$L^*$</th>
<th>$L^R$</th>
<th>$L^T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DCFL</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CFL</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Recursive</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>R.E.</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- $L_1 \cup L_2 =$ Union of $L_1$ and $L_2$
- $L_1 \cap L_2 =$ Intersection of $L_1$ and $L_2$
- $L' =$ Complement of $L$
- $L_1 L_2 =$ Concatenation of $L_1$ and $L_2$
- $L^* =$ Powers of $L$
- $L^R =$ Reverse of $L$
- $L^T =$ Finite transduction of $L$ (may include: intersection/shuffle/perfect-shuffle/quotient with arbitrary regular languages)
Construct DFA for $L_1 \cup L_2$

**Problem**
- Construct a DFA that accepts all strings from the language $L = \{\text{strings with size multiples of 2 or 3}\}$ where $\Sigma = \{a\}$

**Solution**
- Language $L_1 = \{\text{strings with size multiples of 2}\}$
- Language $L_2 = \{\text{strings with size multiples of 3}\}$

[Diagram of DFA with states $p_0$, $p_1$, $q_0$, $q_1$, $q_2$ and transitions for 'a' labeled.]
Construct DFA for $L_1 \cup L_2$

Solution (continued)

- Language $L_1 \cup L_2 = \{\text{strings with size multiples of 2 and 3}\}$
Construct DFA for $L_1 \cup L_2$

<table>
<thead>
<tr>
<th>Union</th>
</tr>
</thead>
</table>
| • Let $M_1$ accept $L_1$, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$  
  Let $M_2$ accept $L_2$, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$  
  • Let $M$ accept $L_1 \cup L_2$, where $M = (Q, \Sigma, \delta, q, F)$. Then  
  $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$  
  $\triangleright$ Cartesian product  
  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)) \ \forall (r_1, r_2) \in Q, a \in \Sigma$  
  $q_0 = (q_1, q_2)$  
  $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$ |
Construct a DFA that accepts all strings from the language
\( L = \{ \text{strings with size multiples of 2 and 3} \} \) where \( \Sigma = \{a\} \)

### Solution

- Language \( L_1 = \{ \text{strings with size multiples of 2} \} \)
- Language \( L_2 = \{ \text{strings with size multiples of 3} \} \)
Construct DFA for $L_1 \cap L_2$

Solution (continued)

- Language $L_1 \cap L_2 = \{\text{strings with size multiples of 2 and 3}\}$
Construct DFA for $L_1 \cap L_2$

<table>
<thead>
<tr>
<th>Intersection</th>
</tr>
</thead>
</table>
| - Let $M_1$ accept $L_1$, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$  
  Let $M_2$ accept $L_2$, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$  
  - Let $M$ accept $L_1 \cap L_2$, where $M = (Q, \Sigma, \delta, q, F)$. Then  
    $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$ ▷ Cartesian product  
    $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)) \quad \forall (r_1, r_2) \in Q, a \in \Sigma$  
    $q_0 = (q_1, q_2)$  
    $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ and } r_2 \in F_2\}$ |
### Problems for practice

Assume $\Sigma = \{a, b\}$ unless otherwise mentioned.

Construct DFA’s for the following languages and generalize:

- $L = \{w \mid |w| = 2\}$
- $L = \{w \mid |w| \leq 2\}$
- $L = \{w \mid |w| \geq 2\}$
- $L = \{w \mid n_a(w) = 2\}$
- $L = \{w \mid n_a(w) \leq 2\}$
- $L = \{w \mid n_a(w) \geq 2\}$
- $L = \{w \mid n_a(w) \mod 3 = 1\}$
- $L = \{w \mid n_a(w) \mod 2 = 0 \text{ and } n_b(w) \mod 2 = 0\}$
- $L = \{w \mid n_a(w) \mod 3 = 2 \text{ and } n_b(w) \mod 2 = 1\}$
- $L = \{w \mid n_a(w) \mod 5 = 3, n_b(w) \mod 3 = 2, \text{ and } n_c(w) \mod 2 = 1\}$ for $\Sigma = \{a, b, c\}$
- $L = \{w \mid n_a(w) \mod 3 \geq n_b(w) \mod 2\}$
### Problems (continued)

- \( L = \{ b \mid \text{binary number } b \mod 3 = 1 \} \) for \( \Sigma = \{0, 1\} \)
- \( L = \{ t \mid \text{ternary number } t \mod 4 = 3 \} \) for \( \Sigma = \{0, 1, 2\} \)
- \( L = \{ w \mid \text{w starts with } a \} \)
- \( L = \{ w \mid \text{w contains } a \} \)
- \( L = \{ w \mid \text{w ends with } a \} \)
- \( L = \{ w \mid \text{w starts with } ab \} \)
- \( L = \{ w \mid \text{w contains } ab \} \)
- \( L = \{ w \mid \text{w ends with } ab \} \)
- \( L = \{ w \mid \text{w starts with } a \text{ and ends with } b \} \)
- \( L = \{ w \mid \text{w starts and ends with different symbols} \} \)
- \( L = \{ w \mid \text{w starts and ends with the same symbol} \} \)
- \( L = \{ w \mid \text{every } a \text{ in } w \text{ is followed by a } b \} \)
- \( L = \{ w \mid \text{every } a \text{ in } w \text{ is never followed by a } b \} \)
### Problems for practice

#### Problems (continued)

- \( L = \{ w \mid \text{every } a \text{ in } w \text{ is followed by } bb \} \)
- \( L = \{ w \mid \text{every } a \text{ in } w \text{ is never followed by } bb \} \)
- \( L = \{ w \mid w = a^m b^n \text{ for } m, n \geq 1 \} \)
- \( L = \{ w \mid w = a^m b^n \text{ for } m, n \geq 0 \} \)
- \( L = \{ w \mid w = a^m b^n c^\ell \text{ for } m, n, \ell \geq 1 \} \text{ for } \Sigma = \{a, b, c\} \)
- \( L = \{ w \mid w = a^m b^n c^\ell \text{ for } m, n, \ell \geq 0 \} \text{ for } \Sigma = \{a, b, c\} \)
- \( L = \{ w \mid \text{second symbol from left end of } w \text{ is } a \} \)
- \( L = \{ w \mid \text{second symbol from right end of } w \text{ is } a \} \)
- \( L = \{ w \mid w = a^3 b x a^3 \text{ such that } x \in \{a, b\}^* \} \)
Two machines or computational models are computationally equivalent if they accept/recognize the same language.
The following models are computationally equivalent: DFA, regular expressions, NFA, and regular grammars.
Closure properties for languages

<table>
<thead>
<tr>
<th>Language</th>
<th>$L_1 \cup L_2$</th>
<th>$L_1 \cap L_2$</th>
<th>$\bar{L}$</th>
<th>$L_1 \circ L_2$</th>
<th>$L^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFA</td>
<td>Easy</td>
<td>Easy</td>
<td>Easy</td>
<td>Hard</td>
<td>Hard</td>
</tr>
<tr>
<td>Regex</td>
<td>Easy</td>
<td>Hard</td>
<td>Hard</td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td>NFA</td>
<td>Easy</td>
<td>Hard</td>
<td>Hard</td>
<td>Easy</td>
<td>Easy</td>
</tr>
</tbody>
</table>

- $L_1 \cup L_2 = \text{Union of } L_1 \text{ and } L_2$
- $L_1 \cap L_2 = \text{Intersection of } L_1 \text{ and } L_2$
- $\bar{L} = \text{Complement of } L$
- $L_1 \circ L_2 = \text{Concatenation of } L_1 \text{ and } L_2$
- $L^* = \text{Powers of } L$
Regular Expressions
Example

- **Arithmetic expression.**
  
  \[(5 + 3) \times 4 = 32 = \text{Number}\]

- **Regular expression.**
  
  \[(a \cup b)a^* = \{a, b, aa, ba, aaa, baa, \ldots\} = \text{Regular language}\]

Application

- **Regular expressions in Linux.**
  
  Used to find patterns in filenames, file content etc.
  Used in Linux tools such as awk, grep, and Perl.
  
### Definition

- The following are regular expressions.  
  \( \epsilon, \phi, a \in \Sigma \).
- If \( R_1 \) and \( R_2 \) are regular expressions, then the following are regular expressions.
  - (Union.)  
    \( R_1 \cup R_2 \)
  - (Concatenation.)  
    \( R_1 \circ R_2 \)
  - (Kleene star.)  
    \( R_1^* \)
### Examples

<table>
<thead>
<tr>
<th>Regular language</th>
<th>Regular expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>( \phi )</td>
</tr>
<tr>
<td>{(\epsilon)}</td>
<td>( \epsilon )</td>
</tr>
<tr>
<td>{a}</td>
<td>a</td>
</tr>
<tr>
<td>{a, b}</td>
<td>( a \cup b )</td>
</tr>
<tr>
<td>{a}{b}</td>
<td>ab</td>
</tr>
<tr>
<td>{a}*</td>
<td>a*</td>
</tr>
<tr>
<td>{aab}*{a, ab}</td>
<td>(aab)*(a \cup ab)</td>
</tr>
<tr>
<td>({aa, bb} \cup {a, b}{aa}*{ab, ba})*</td>
<td>(aa \cup bb \cup (a \cup b)(aa)<em>(ab \cup ba))</em></td>
</tr>
</tbody>
</table>

### Equality

- Two regular expressions are equal if they describe the same regular language. E.g.:

  \((a^*b^*)^* = (a \cup b)^*ab(a \cup b)^* \cup b^*a^* = (a \cup b)^* = \Sigma^*\)
Examples

Let $\Sigma = a \cup b$, $R^+ = RR^*$, and $R^k = \underbrace{R \cdots R}_{k \text{ times}}$

- $L = \{w \mid |w| = 2\}$
  $R = \Sigma \Sigma$
- $L = \{w \mid |w| \leq 2\}$
  $R = \epsilon \cup \Sigma \cup \Sigma \Sigma$
- $L = \{w \mid |w| \geq 2\}$
  $R = \Sigma \Sigma \Sigma^*$
- $L = \{w \mid n_a(w) = 2\}$
  $R = b^* ab^* ab^*$
- $L = \{w \mid n_a(w) \leq 2\}$
  $R = b^* \cup b^* ab^* \cup b^* ab^* ab^*$
- $L = \{w \mid n_a(w) \geq 2\}$
  $R = b^* ab^* ab^* (ab^*)^*$
### Rules

**Beware of $\phi$ and $\epsilon$**

Suppose $R$ is a regular expression.

- $R \cup \phi = R$
- $R \circ \epsilon = R$
- $R \cup \epsilon$ may not equal $R$
  
  (e.g.: $R = 0$, $L(R) = \{0\}$, $L(R \cup \epsilon) = \{0, \epsilon\}$)
- $R \circ \phi$ may not equal $R$
  
  (e.g.: $R = 0$, $L(R) = \{0\}$, $L(R \circ \phi) = \phi$)
**Rules**

Suppose $R_1, R_2, R_3$ are regular expressions. Then

- $R_1 \phi = \phi R_1 = \phi$
- $R_1 \epsilon = \epsilon R_1 = R_1 \cup \phi = \phi \cup R_1 = R_1$
- $R_1 \cup R_1 = R_1$
- $R_1 \cup R_2 = R_2 \cup R_1$
- $R_1 (R_2 \cup R_3) = R_1 R_2 \cup R_1 R_3$
- $(R_1 \cup R_2) R_3 = R_1 R_3 \cup R_2 R_3$
- $R_1 (R_2 R_3) = (R_1 R_2) R_3$
- $\phi^* = \epsilon$
- $(\epsilon \cup R_1)^* = (\epsilon \cup R_1)^+ = R_1^*$
- $R_1^* (\epsilon \cup R_1) = (\epsilon \cup R_1) R_1^* = R_1^*$
- $R_1^* R_2 \cup R_2 = R_1^* R_2$
- $R_1 (R_2 R_1)^* = (R_1 R_2)^* R_1$
- $(R_1 \cup R_2)^* = (R_1 * R_2)^* R_1^* = (R_2^* R_1)^* R_2^*$
Construct a regex for $\Sigma = \{a, b\}$

### Problem

- Construct a regular expression to describe the language $L = \{w \mid n_a(w) \text{ is odd}\}$
**Problem**

- Construct a regular expression to describe the language 
  \[ L = \{ w \mid n_a(w) \text{ is odd} \} \]

**Solution**

- **Incorrect expressions.**
  
  \[ b^*ab^*(ab^*a)^*b^* \]
  \[ b^*a(b^*ab^*ab^*)^* \]

- **Why?**

- **Correct expressions.**

  \[ b^*ab^*(b^*ab^*ab^*)^* \]
  \[ b^*ab^*(ab^*ab^*)^* \]
  \[ b^*a(b^*ab^*a)^*b^* \]
  \[ b^*a(b \cup ab^*a)^* \]
  \[ (b \cup ab^*a)^*ab^* \]

- **Why?**
Problem

• Construct a regular expression to describe the language $L = \{ w \mid w \text{ ends with } b \text{ and does not contain } aa \}$
### Problem

- Construct a regular expression to describe the language $L = \{w \mid w \text{ ends with } b \text{ and does not contain } aa\}$

### Solution

- A string not containing $aa$ means that every $a$ in the string:
  - is immediately followed by $b$, or
  - is the last symbol of the string
- Each string in the language has to end with $b$.
- Hence, every $a$ in each string of the language is immediately followed by $b$
- Regular expression is: $(b \cup ab)^+$
Construct a regex to recognize identifiers in C

Problem

- **Identifiers** are the names you supply for variables, types, functions, and labels.
- Construct a regular expression to recognize the **identifiers** in the C programming language i.e., \( L = \{ \text{identifiers in C} \} \)
# Construct a regex to recognize identifiers in C

## Problem

- **Identifiers** are the names you supply for variables, types, functions, and labels.
- Construct a regular expression to recognize the **identifiers** in the C programming language i.e., $L = \{\text{identifiers in C}\}$

## Solution

- C identifier = FirstLetter OtherLetters
  - FirstLetter = English letter or underscore
  - OtherLetters = Alphanumeric letters or underscore
- Let $L = \{a, \ldots, z, A, \ldots, Z\}$ and $D = \{0, 1, \ldots, 9\}$
- Regular expression is:
  \[ R = \text{FirstLetter} \circ \text{OtherLetters} \]
  - FirstLetter = $(L \cup _)$
  - OtherLetters = $(L \cup D \cup _)$
Construct a regex to recognize decimals in C

Problem

• Construct a regular expression to recognize the decimal numbers in the C programming language i.e.,
  \( L = \{ \text{decimal numbers in C} \} \)

• Examples: 14, +1, −12, 14.3, −.99, 16., 3E14, −1.00E2, 4.1E−1, and .3E + 2
### Problem
- Construct a regular expression to recognize the decimal numbers in the C programming language i.e.,
  \[ L = \{ \text{decimal numbers in C} \} \]
- Examples: 14, +1, −12, 14.3, −.99, 16., 3E14, −1.00E2, 4.1E−1, and .3E + 2

### Solution
- C decimal number = Sign Decimals Exponent
- Let \( D = \{ 0, 1, \ldots, 9 \} \)
- Regular expression is:
  \[
  R = \text{Sign} \circ \text{Decimals} \circ \text{Exponent}
  \]
  \[
  \text{Sign} = (+ \cup - \cup \epsilon)
  \]
  \[
  \text{Decimals} = (D^+ \cup D^+.D^* \cup D^*.D^+) \]
  \[
  \text{Exponent} = (\epsilon \cup E \text{ Sign } D^+)
  \]
Nondeterministic Finite Automata (NFA)
Example NFA’s

Examples

<table>
<thead>
<tr>
<th>Difference</th>
<th>DFA</th>
<th>NFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple transitions</td>
<td>1 exiting arrow</td>
<td>≥ 0 exiting arrows</td>
</tr>
<tr>
<td>Epsilon transitions</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>Missing transitions</td>
<td>No missing transitions</td>
<td>Missing transitions mean transitions to sink/reject state</td>
</tr>
</tbody>
</table>
### Intuition

Nondeterministic computation = Parallel computation
(NFA searches all possible paths in a graph to the accept state)

- When NFA has multiple choices for the same input symbol, think of it as a process forking multiple processes for parallel computation.
- A string is accepted if any of the parallel processes accepts the string.

Nondeterministic computation = Tree of possibilities
(NFA magically guesses a right path to the accept state)

- Root of the tree is the start of the computation.
- Every branching point is the decision-making point consisting of multiple choices.
- Machine accepts a string if any of the paths from the root of the tree to a leaf leads to an accept state.
Why care for NFA’s?

Uses of NFA’s

- Constructing NFA’s is easier than directly constructing DFA’s for many problems.
  Hence, construct NFA’s and then convert them to DFA’s.
- NFA’s are easier to understand than DFA’s.
Construct NFA for $\Sigma = \{0, 1\}$

**Problem**
- Construct a NFA that accepts all strings from the language $L = \{\text{strings containing 11 or 101}\}$

**Solution**

How does the machine work for the input 010110?
- What is the equivalent DFA for solving the problem?
Construct NFA for $\Sigma = \{0, 1\}$

Source: Anil Maheshwari and Michiel Smid’s Theory of Computation
Problem

- Construct a NFA that accepts all strings from the language $L = \{\text{strings of size multiples of 2 or 3 or 5}\}$

Solution

What is the equivalent DFA for solving the problem?
What is a nondeterministic finite automaton (NFA)?

- Nondeterministic = Event paths cannot be determined precisely
- Finite = Finite and small amount of space used
- Automaton = Computing machine

**Definition**

A **nondeterministic finite automaton (NFA)** $M$ is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where,

1. $Q$: A finite set (**set of states**).  ▶ **Space (computer memory)**
2. $\Sigma$: A finite set (**alphabet**).
3. $\delta: Q \times (\Sigma \cup \epsilon) \rightarrow P(Q)$ is the transition function, where $P(Q)$ is the power set of $Q$.  ▶ **Time (computation)**
4. $q_0$: The **start state** (belongs to $Q$).
5. $F$: The set of **accepting/final states**, where $F \in Q$. 
Problem

- Convert the NFA to a DFA.

Source: Anil Maheshwari and Michiel Smid’s Theory of Computation
Construct DFA for the given NFA

Solution

• NFA $M$ is specified as
  
  Set of states is $Q = \{1, 2, 3\}$
  Set of symbols is $\Sigma = \{a, b\}$
  Start state is 1
  Set of accept states is $F = \{1\}$
  Transition function $\delta$ is:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{3}</td>
<td>$\phi$</td>
<td>{2}</td>
</tr>
<tr>
<td>2</td>
<td>{1}</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>3</td>
<td>{2}</td>
<td>{2, 3}</td>
<td>$\phi$</td>
</tr>
</tbody>
</table>

• How do you convert this NFA to DFA?
Construct DFA for the given NFA

Solution

- NFA $M$ is specified as
  - Set of states is $Q = \{1, 2, 3\}$
  - Set of symbols is $\Sigma = \{a, b\}$
  - Start state is 1
  - Set of accept states is $F = \{1\}$
  - Transition function $\delta$ is:

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$a$</th>
<th>$b$</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${3}$</td>
<td>$\phi$</td>
<td>${2}$</td>
</tr>
<tr>
<td>2</td>
<td>${1}$</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>3</td>
<td>${2}$</td>
<td>${2, 3}$</td>
<td>$\phi$</td>
</tr>
</tbody>
</table>

- How do you convert this NFA to DFA?
  If NFA has states $Q$, then construct a DFA with states $P(Q)$. 
Construct DFA for the given NFA

<table>
<thead>
<tr>
<th>Solution (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi \xrightarrow{a} \phi )</td>
</tr>
<tr>
<td>( \phi \xrightarrow{b} \phi )</td>
</tr>
<tr>
<td>( {1} \xrightarrow{a} {3} )</td>
</tr>
<tr>
<td>( {1} \xrightarrow{b} \phi )</td>
</tr>
<tr>
<td>( {2} \xrightarrow{a} {1, 2} )</td>
</tr>
<tr>
<td>( {2} \xrightarrow{b} \phi )</td>
</tr>
<tr>
<td>( {3} \xrightarrow{a} {2} )</td>
</tr>
<tr>
<td>( {3} \xrightarrow{b} {2, 3} )</td>
</tr>
</tbody>
</table>
Construct DFA for the given NFA

Solution (continued)

Source: Anil Maheshwari and Michiel Smid’s Theory of Computation
Construct DFA for the given NFA

Solution (continued)

Source: Anil Maheshwari and Michiel Smid’s Theory of Computation
Construct DFA for the given NFA

Convert NFA to DFA

• Let $N = (Q, \Sigma, \delta, q, F)$ be the NFA.
  Let $M = (Q', \Sigma, \delta', q', F')$ be the DFA. Then
• $Q' = P(Q)$ \quad \triangleright \text{Power set of } Q$
  $q' = C_\epsilon(\{q\})$ \quad \triangleright \text{ } \epsilon\text{-closure of the start state}$
  $F' = \{S \in Q' \mid S \cap F \neq \phi\}$ \quad \triangleright \text{ } S \cap F \neq \phi \text{ means that } S$
  contains at least one accept state of $N$
  $\delta' : Q' \times \Sigma \rightarrow Q'$ is defined as follows:
  For all state $S \in Q'$ and for all letter $a \in \Sigma,$
  \[
  \delta'(S, a) = \bigcup_{s \in S} C_\epsilon(\delta(s, a))
  \]
Union of NFA

Source: Margaret Fleck and Sariel Har-Peled's Notes on Theory of Computation
Concatenation of NFA

Source: Margaret Fleck and Sariel Har-Peled's Notes on Theory of Computation
Star of NFA

Source: Margaret Fleck and Sariel Har-Peled’s Notes on Theory of Computation
Construct a NFA for \((aa \cup aab)^*b\)

Problem

- Construct a NFA for the regular expression \((aa \cup aab)^*b\).
Construct a NFA for \((aa \cup aab)^*b\)

**Problem**

- Construct a NFA for the regular expression \((aa \cup aab)^*b\).

**Solution**

Source: John Martin's Introduction to Languages and the Theory of Computation.
Problem

- Construct a NFA for the regular expression $(aab)^*(a \cup aba)^*$. 

Source: John Martin's Introduction to Languages and the Theory of Computation.
Construct a NFA for \((aab)^*(a \cup aba)^*\)

**Problem**

- Construct a NFA for the regular expression \((aab)^*(a \cup aba)^*\).

**Solution**

![Diagram of NFA](image)

Source: John Martin’s Introduction to Languages and the Theory of Computation.
Non-Regular Languages
### Regular or non-regular languages

**Problems**

Let $\Sigma = \{a, b\}$ unless mentioned otherwise. Check if the languages are regular or non-regular ($X$):

- $L = \{w \mid w = a^n \text{ and } n \leq 10^{100}\}$
- $L = \{w \mid w = a^n \text{ and } n \geq 1\}$
- $L = \{w \mid w = a^m b^n \text{ and } m, n \geq 1\}$
- $L = \{w \mid w = a^* b^*\}$
- $L = \{w \mid w = a^n b^n \text{ and } n \geq 1\}$
- $L = \{ww^R \mid |w| = 3\}$
- $L = \{ww^R \mid |w| \geq 1\}$
- $L = \{w \mid w = w^R \text{ and } |w| \geq 1\}$
- $L = \{w \mid w = a^{2i+1}b^{3j+2} \text{ and } i, j \geq 1\}$
- $L = \{w \mid w = a^n \text{ and } n \text{ is a square}\}$
- $L = \{w \mid w = a^n \text{ and } n \text{ is a prime}\}$
- $L = \{w \mid w = a^i b^{j^2} \text{ and } i, j \geq 1\}$
### Regular or non-regular languages

#### Problems

Let $\Sigma = \{a, b\}$ unless mentioned otherwise. Check if the languages are regular or non-regular (X):

- $L = \{w \mid w = a^n \text{ and } n \leq 10^{100}\}$
- $L = \{w \mid w = a^n \text{ and } n \geq 1\}$
- $L = \{w \mid w = a^m b^n \text{ and } m, n \geq 1\}$
- $L = \{w \mid w = a^* b^*\}$
- $L = \{w \mid w = a^n b^n \text{ and } n \geq 1\}$
- $L = \{ww^R \mid |w| = 3\}$
- $L = \{ww^R \mid |w| \geq 1\}$
- $L = \{w \mid w = w^R \text{ and } |w| \geq 1\}$
- $L = \{w \mid w = a^{2i+1} b^{3j+2} \text{ and } i, j \geq 1\}$
- $L = \{w \mid w = a^n \text{ and } n \text{ is a square}\}$
- $L = \{w \mid w = a^n \text{ and } n \text{ is a prime}\}$
- $L = \{w \mid w = a^i b^{j^2} \text{ and } i, j \geq 1\}$
Regular or non-regular languages

Problems (continued)

- $L = \{ w \mid n_a(w) = n_b(w) \}$
- $L = \{ w \mid n_a(w) \mod 3 \geq n_b(w) \mod 5 \}$
- $L = \{ w \mid w = a^i b^j \text{ and } j > i \geq 1 \}$
- $L = \{ w x w^R \mid x \in \Sigma^*, |w|, |x| \geq 1, \text{ and } |x| \leq 5 \}$
- $L = \{ w x w^R \mid x \in \Sigma^* \text{ and } |w|, |x| \geq 1 \}$
- $L = \{ x w w^R y \mid x, y \in \Sigma^* \text{ and } |w|, |x|, |y| \geq 1 \}$
- $L = \{ x w w^R \mid x \in \Sigma^* \text{ and } |w|, |x| \geq 1 \}$
- $L = \{ w w^R y \mid y \in \Sigma^* \text{ and } |w|, |y| \geq 1 \}$
### Problems (continued)

- \( L = \{ w \mid n_a(w) = n_b(w) \} \) .................................................. \( \times \)
- \( L = \{ w \mid n_a(w) \mod 3 \geq n_b(w) \mod 5 \} \)
- \( L = \{ w \mid w = a^i b^j \text{ and } j > i \geq 1 \} \) ................................\( \times \)
- \( L = \{ wxw^R \mid x \in \Sigma^*, |w|, |x| \geq 1, \text{ and } |x| \leq 5 \} \) ............. \( \times \)
- \( L = \{ wxw^R \mid x \in \Sigma^* \text{ and } |w|, |x| \geq 1 \} \)
- \( L = \{ xww^Ry \mid x, y \in \Sigma^* \text{ and } |w|, |x|, |y| \geq 1 \} \)
- \( L = \{ xww^R \mid x \in \Sigma^* \text{ and } |w|, |x| \geq 1 \} \) ................................\( \times \)
- \( L = \{ ww^Ry \mid y \in \Sigma^* \text{ and } |w|, |y| \geq 1 \} \) ................................\( \times \)
How to prove that certain languages are not regular?

<table>
<thead>
<tr>
<th>Pumping lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Many languages are not regular.</td>
</tr>
<tr>
<td>• <strong>Pumping lemma</strong> is a method to prove that certain languages are not regular.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pumping property</th>
</tr>
</thead>
<tbody>
<tr>
<td>• If a language is regular, then it must have the <em>pumping property</em>.</td>
</tr>
<tr>
<td>• If a language does not have the pumping property, then the language is not regular.  ▶ Proof by contraposition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How to prove languages non-regular using pumping lemma?</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>Proof by contradiction.</strong></td>
</tr>
<tr>
<td>Assume that the language is regular.</td>
</tr>
<tr>
<td>Show that the language does not have the pumping property.</td>
</tr>
<tr>
<td>Contradiction! Hence, the language has to be non-regular.</td>
</tr>
</tbody>
</table>
Pumping property of regular languages

- Suppose a DFA $M$ with $p$ number of states accepts a very long string $s \geq p$ from a language $L$.
- From pigeonhole principle, at least one state is visited twice.
- This implies that the string went through a loop.
Pumping property of regular languages

Observations

- Suppose string $s$ has more characters than the number of states in the DFA, i.e., $|s| \geq p$
- String $s$ can be split into three parts i.e., $s = xyz$ where
  - $x$: string before the first loop
  - $y$: string of the first loop
  - $z$: string after the first loop (might contain loops)
- Loop must appear i.e., $|y| \geq 1$
  ($x$ and $z$ can be empty)
- Loop must appear in the first $p$ characters of $s$ i.e., $|xy| \leq p$
Pumping property of regular languages

**Idea**

- An infinite number of strings can be pumped with loop length and they must also be in the language.
- Formally, for all $i \geq 0$, $xy^iz$ must be in the language.
- $xz, xyz, xyyz, xyyyz$, etc must also belong to the language.
Pumping lemma for regular languages

Theorem

Suppose $L$ is a language over alphabet $\Sigma$. Suppose $L$ is accepted by a finite automaton $M$ having $p$ states. Then, every long string $w \in L$ satisfying $|w| \geq p$ can be split into three strings $w = xyz$ such that the following three conditions are true.

- $|xy| \leq p$.
- $|y| \geq 1$.
- For every $i \geq 0$, the string $xy^iz$ also belongs to $L$. 

Problem

- Prove that $L = \{a^n b^n \mid n \geq 0\}$ is not a regular language.
**Problem**

- Prove that $L = \{a^n b^n \mid n \geq 0\}$ is not a regular language.

**Solution**

- Suppose $L$ is regular. Then it must satisfy pumping property.
- Suppose $w = a^p b^p$.
- Let $w = xyz = \framebox{a^{p-k}} \framebox{a^k} \framebox{b^p}$ where $k \in [1, p]$.
- We have $|xy| \leq p$ and $|y| \geq 1$.
- Also, $xy^i z$ must belong to $L$ for all $i \geq 0$.
- But, $xyyz$ is not in $L$.
  - Reason: $xyyz = a^{p-k} a^k a^k b^p = a^{p+k} b^p \not\in L$.
  - $xyyz$ has more $a$'s than $b$'s.
- Contradiction! Hence, $L$ is not regular.
Problem

Prove that $L = \{w \mid n_a(w) = n_b(w)\}$ is not a regular language.
Problem

- Prove that $L = \{ w \mid n_a(w) = n_b(w) \}$ is not a regular language.

Solution

- Suppose $L$ is regular. Then it must satisfy pumping property.
- Suppose $w = (ab)^p$.
- Let $w = xyz = \epsilon (ab)^1 (ab)^{p-1}$
- We have $|xy| \leq p$ and $|y| \geq 1$.
- Also, $xy^i z$ must belong to $L$ for all $i \geq 0$.
- $xy^i z$ belongs to $L$ for all $i \geq 0$.
- No contradiction! Hence, $L$ is regular.

Incorrect solution! Why?
1. If we cannot find a contradiction, that does not prove anything.
2. We must try for all possible values of $y$ such that $|xy| \leq p$.
3. The chosen string $(ab)^p$ is a bad string to work on.
**Problem**

- Prove that \( L = \{ w \mid n_a(w) = n_b(w) \} \) is not a regular language.

**Solution**

- Suppose \( L \) is regular. Then it must satisfy pumping property.
- Suppose \( w = (ab)^p \).
- Let \( w = xyz = \epsilon (ab)^1 (ab)^{p-1} \).
- We have \( |xy| \leq p \) and \( |y| \geq 1 \).
- Also, \( xy^iz \) must belong to \( L \) for all \( i \geq 0 \).
- \( xy^iz \) belongs to \( L \) for all \( i \geq 0 \).
- No contradiction! Hence, \( L \) is regular.

**Mistakes**

**Incorrect solution!** Why? Multiple reasons:

1. If we cannot find a contradiction, that does not prove anything.
2. We must try for all possible values of \( y \) such that \( |xy| \leq p \).
3. The chosen string \( (ab)^p \) is a bad string to work on.
$L = \{ w \mid n_a(w) = n_b(w) \}$ is non-regular

### Problem

- Prove that $L = \{ w \mid n_a(w) = n_b(w) \}$ is not a regular language.

### Solution

- Suppose $L$ is regular. Then it must satisfy pumping property.
- Suppose $w = a^p b^p$.
- Let $w = xyz = a^{p-k} a^k b^p$ where $k \in [1, p]$.
- We have $|xy| \leq p$ and $|y| \geq 1$.
- Also, $x y^i z$ must belong to $L$ for all $i \geq 0$.
- But, $x y y z$ is not in $L$.
  - Reason: $x y y z = a^{p-k} a^k a^k b^p = a^{p+k} b^p \not\in L$.
  - $x y y z$ has more $a$’s than $b$’s.
- Contradiction! Hence, $L$ is not regular.

**Takeaway**

1. If a subset of language $L$ is not regular, then $L$ is not regular.
2. Reduction! Reduce a problem to another. Reuse its solution.
Problem

- Prove that $L = \{w \mid n_a(w) = n_b(w)\}$ is not a regular language.

Solution

- Suppose $L$ is regular. Then it must satisfy pumping property.
- Suppose $w = a^p b^p$.
- Let $w = xyz = a^{p-k}a^k b^p$ where $k \in [1, p]$.
- We have $|xy| \leq p$ and $|y| \geq 1$.
- Also, $xy^iz$ must belong to $L$ for all $i \geq 0$.
- But, $xyyz$ is not in $L$.
  Reason: $xyyz = a^{p-k}a^k a^k b^p = a^{p+k} b^p \notin L$.
  $xyyz$ has more $a$'s than $b$'s.
- Contradiction! Hence, $L$ is not regular.

Takeaway

1. If a subset of language $L$ is not regular, then $L$ is not regular.
2. **Reduction!** Reduce a problem to another. Reuse its solution.
Problem

• Prove that $L = \{w \mid n_a(w) = n_b(w)\}$ is not a regular language.

Solution (without using pumping lemma)

• Suppose $L$ is regular.
• We know that $L' = \{w \mid w = a^i b^j \text{ and } i, j \geq 0\}$.
• As regular languages are closed under intersection, $L \cap L'$ must also be regular.
• We see that $L \cap L = \{w \mid w = a^n b^n \text{ and } n \geq 0\}$.
• But, this language was earlier proved to be non-regular.
• Contradiction! Hence, $L$ is not regular.