Languages: What is Computing?

A Formal Model of a Computing Problem
Decision Problems and Languages
Describing a Language: Regular Expressions
Complexity of a Computing Problem
Comparing infinite sets.

Countable.
- $\mathbb{N}_0, E, \mathbb{Z}, \mathbb{Q}$ are countable.
- Finite binary strings $B$ is countable.

Uncountable
- *Infinite* binary strings are uncountable.
- Reals are uncountable.

Infinity and computing.
- Programs are finite binary strings (countable).
- Functions we might like to compute are infinite binary strings (uncountable).
- Conclusion: there are **MANY** functions which *cannot* be computed by programs.
Decision problems.

Languages.
  - Describing a language.

Complexity of a computing problem.
What is a Computing Problem?

*Decide*  **YES** or **NO**  whether a given integer $n \in \mathbb{N}$ is prime.
What is a Computing Problem?

Decide \text{YES} or \text{NO} whether a given integer $n \in \mathbb{N}$ is prime.

List the primes in increasing order (primes are countable),

$$\text{primes} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \ldots\}$$
**What is a Computing Problem?**

*Decide*  

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
</table>

whether a given integer \( n \in \mathbb{N} \) is prime.

List the primes in increasing order (primes are countable),

\[
\text{primes} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \ldots\}
\]

Given \( n \in \mathbb{N} \), walk through this list.

1. If you come to \( n \) output **YES**.  
2. If you come to a number bigger than \( n \), output **NO**.

Not the smartest approach to primality testing, but gets to the heart of computing
What is a Computing Problem?

*Decide* \(\text{YES}\) or \(\text{NO}\) whether a given integer \(n \in \mathbb{N}\) is prime.

List the primes in increasing order (primes are countable),

\[\text{primes} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \ldots\}\]

Given \(n \in \mathbb{N}\), walk through this list.

1. If you come to \(n\) output \(\text{YES}\).
2. If you come to a number bigger than \(n\), output \(\text{NO}\).

Not the smartest approach to primality testing, but gets to the heart of computing...
\[ \mathcal{L}_{\text{prime}} = \{10, 11, 101, 111, 1011, 1101, 10001, 10011, 10111, 11101, \ldots \}. \]  
(Primes in binary)
$L_{\text{prime}} = \{10, 11, 101, 111, 1011, 1101, 10001, 10011, 10111, 11101, \ldots \}$. (primes in binary)

9 is prime $\iff$ the string 1001 is in $L_{\text{prime}}$. 
Decision Problems

\[ \mathcal{L}_{\text{prime}} = \{10, 11, 101, 111, 1011, 1101, 1011, 11101, \ldots \} \]  
(primes in binary)

9 is prime \(\iff\) the string 1001 is in \(\mathcal{L}_{\text{prime}}\).

The light is off. Every push toggles between on and off. Given the number of pushes, decide whether the light is on or off. Encode number of pushes by a binary string, e.g. 101 means 5 pushes.
Decision Problems

\[ \mathcal{L}_{\text{prime}} = \{10, 11, 101, 111, 1011, 1101, 10001, 10011, 10111, 11011, \ldots \}. \]  

(primes in binary)

9 is prime \(\leftrightarrow\) the string 1001 is in \(\mathcal{L}_{\text{prime}}\).

The light is off. Every push toggles between on and off.  
Given the number of pushes, decide whether the light is on or off.  
Encode number of pushes by a binary string, e.g. 101 means 5 pushes.

\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \}. \]

The light is on for 1010 pushes, if and only if 1010 \(\in\) \(\mathcal{L}_{\text{push}}\).
Decision Problems

$$L_{\text{prime}} = \{10, 11, 101, 111, 1011, 1101, 10001, 10011, 10111, 11101, \ldots \}.$$  

(Primes in binary)

9 is prime $\iff$ the string 1001 is in $L_{\text{prime}}$.

The light is off. Every push toggles between on and off. Given the number of pushes, decide whether the light is on or off. Encode number of pushes by a binary string, e.g. 101 means 5 pushes.

$$L_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \}.$$  

The light is on for 1010 pushes, if and only if 1010 $\in L_{\text{push}}$.

The door should open if a person is on the mat. Walk on (1) or off (0). E.g. 10110 means on, off, on, on, off $\rightarrow$ open.
Decision Problems

\[ L_{\text{prime}} = \{10, 11, 101, 111, 1011, 1101, 10001, 10011, 10111, 11101, \ldots \}. \] (primes in binary)

9 is prime \( \leftrightarrow \) the string 1001 is in \( L_{\text{prime}} \).

The light is off. Every push toggles between on and off.
Given the number of pushes, decide whether the light is on or off.
Encode number of pushes by a binary string, e.g. 101 means 5 pushes.

\[ L_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \}. \]

The light is on for 1010 pushes, if and only if 1010 \( \in L_{\text{push}} \).

The door should open if a person is on the mat.
Walk on (1) or off (0). E.g. 10110 means on, off, on, on, off \( \rightarrow \) open.

\[ L_{\text{door}} = \{1, 11, 101, 110, 111, 1011, 1101, 1110, 1111, \ldots \}. \]

Given input \( w \), e.g. \( w = 1011 \), the door is open if and only if \( w \in L_{\text{door}} \).
Decision Problems

\[ \mathcal{L}_{\text{prime}} = \{10, 11, 101, 111, 1011, 1101, 10001, 10011, 10111, 11011, \ldots \} \]. (primes in binary)

9 is prime \( \leftrightarrow \) the string 1001 is in \( \mathcal{L}_{\text{prime}} \).

The light is off. Every push toggles between on and off.
Given the number of pushes, decide whether the light is on or off.
Encode number of pushes by a binary string, e.g. 101 means 5 pushes.

\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \].

The light is on for 1010 pushes, if and only if 1010 \( \in \mathcal{L}_{\text{push}} \).

The door should open if a person is on the mat.
Walk on (1) or off (0). E.g. 10110 means on, off, on, on, off \( \rightarrow \) open.

\[ \mathcal{L}_{\text{door}} = \{1, 11, 101, 110, 111, 1011, 1101, 1110, 1111, \ldots \} \].

Given input \( w \), e.g. \( w = 1011 \), the door is open if and only if \( w \in \mathcal{L}_{\text{door}} \).

Decision problems can be formulated as testing membership in a set of strings
(a) \textbf{Optimization} What’s distance between nodes ① and ③? Answer: 2
A Decision Problem on Graphs

(a) [Optimization] What’s distance between nodes 1 and 3? Answer: 2
(b) [Decision] Is there a path between 1 and 3 of length at most 3? YES.
(a) [Optimization] What’s distance between nodes ① and ③? Answer: 2
(b) [Decision] Is there a path between ① and ③ of length at most 3? Yes.

(a) is harder than (b): (a)’s answer gives (b)’s answer instantly.
A Decision Problem on Graphs

(a) [Optimization] What’s distance between nodes $\odot$ and $\odot$? Answer: $2$

(b) [Decision] Is there a path between $\odot$ and $\odot$ of length at most 3? $\text{YES}$.

(a) is harder than (b): (a)’s answer gives (b)’s answer instantly.

Let’s encode (b) as a string identifying the graph, nodes of interest and target distance.
A Decision Problem on Graphs

(a) [Optimization] What’s distance between nodes 1 and 3? Answer: 2

(b) [Decision] Is there a path between 1 and 3 of length at most 3? Yes.

(a) is harder than (b): (a)’s answer gives (b)’s answer instantly.

Let’s encode (b) as a string identifying the graph, nodes of interest and target distance.

“Is there a path of length at most 3 between nodes 1 and 3 in the graph above.”
(a)[Optimization] What’s distance between nodes ① and ③? Answer: 2
(b)[Decision] Is there a path between ① and ③ of length at most 3? **YES.**

(a) is harder than (b): (a)’s answer gives (b)’s answer instantly.

Let’s *encode* (b) as a string identifying the graph, nodes of interest and target distance.

“Is there a path of length at most 3 between nodes ① and ③ in the graph above.” becomes

“ 1, 2, 3, 4 ”

nodes
(a) **Optimization** What’s distance between nodes 1 and 3? Answer: 2
(b) **Decision** Is there a path between 1 and 3 of length at most 3? **YES.**

(a) is harder than (b): (a)’s answer gives (b)’s answer instantly.

Let’s *encode* (b) as a string identifying the graph, nodes of interest and target distance.

“Is there a path of length at most 3 between nodes 1 and 3 in the graph above.”

becomes

“ 1, 2, 3, 4 | (1, 2)(2, 3)(3, 4)(4, 1) ”

Creator: Malik Magdon-Ismail
Languages: What is Computing?: 6 / 17
A Decision Problem on Graphs

(a) **Optimization** What’s distance between nodes 1 and 3? Answer: 2

(b) **Decision** Is there a path between 1 and 3 of length at most 3? **YES.**

(a) is harder than (b): (a)’s answer gives (b)’s answer instantly.

Let’s *encode* (b) as a string identifying the graph, nodes of interest and target distance.

“Is there a path of length at most 3 between nodes 1 and 3 in the graph above.”

becomes

“ 1, 2, 3, 4 | (1, 2)(2, 3)(3, 4)(4, 1) | 1, 3  ”

Creator: Malik Magdon-Ismail
A Decision Problem on Graphs

(a) **[Optimization]** What’s distance between nodes 1 and 3? Answer: 2

(b) **[Decision]** Is there a path between 1 and 3 of length at most 3? **YES.**

(a) is harder than (b): (a)’s answer gives (b)’s answer instantly.

Let’s *encode* (b) as a string identifying the graph, nodes of interest and target distance.

“Is there a path of length at most 3 between nodes 1 and 3 in the graph above.”

becomes

```
“ 1, 2, 3, 4 | (1, 2)(2, 3)(3, 4)(4, 1) | 1, 3 | 3 ”
```

The graph problem can be encoded as a binary string using ASCII

```
001100010011000011001000110110000110011001011000011001100101100001100000110100001100000110100100101010000110000110010001100000110010001
```

Creator: Malik Magdon-Ismail
Languages: What is Computing?: 6 / 17
Decision is Harder than Optimization →
A Decision Problem on Graphs

(a) [Optimization] What’s distance between nodes 1 and 3? Answer: 2
(b) [Decision] Is there a path between 1 and 3 of length at most 3? YES.

(a) is harder than (b): (a)’s answer gives (b)’s answer instantly.

Let’s encode (b) as a string identifying the graph, nodes of interest and target distance.

“Is there a path of length at most 3 between nodes 1 and 3 in the graph above.”

becomes

“ 1, 2, 3, 4 | (1, 2)(2, 3)(3, 4)(4, 1) | 1, 3 | 3 ”

The graph problem can be encoded as a binary string using ASCII

0011001001011000011001000101100001100111111000010110001100100101100011001000101100011001000101100011000001100100010110000110100001100110011001100110111110000110100000110001001011000011010000101001001010000011001000101100001100010010100101111100001100010010110000110011

\[ \mathcal{L}_\text{path} = \{ \text{All strings of the form “nodes | edges | endpoints of path | target distance” for which } \}
\{ \text{the distance between the endpoints in the graph is at most the target distance. } \} \]

Pop Quiz. YES or NO: “ 1, 2, 3, 4, 5 | (1, 2)(2, 3)(3, 5)(3, 4) | 1, 5 | 2 ”
Is Optimization Really Harder than Decision?
Is Optimization Really Harder than Decision?

If you can solve the decision problem, you can solve the optimization problem.

Is there a path in the graph between nodes $\times$ and $\triangleright$ of length at most 1? [NO]
Is Optimization Really Harder than Decision?

If you can solve the decision problem, you can solve the optimization problem.

Is there a path in the graph between nodes ⊙ and ⊘ of length at most 1? \( \text{NO} \)
Is there a path in the graph between nodes ⊙ and ⊘ of length at most 2? \( \text{NO} \)
Is Optimization Really Harder than Decision?

If you can solve the decision problem, you can solve the optimization problem.

| Is there a path in the graph between nodes $\times$ and $\circ$ of length at most $1$? | NO |
| Is there a path in the graph between nodes $\times$ and $\circ$ of length at most $2$? | NO |
| Is there a path in the graph between nodes $\times$ and $\circ$ of length at most $3$? | NO |
| Is there a path in the graph between nodes $\times$ and $\circ$ of length at most $4$? | YES |

You ask the decision question until the answer is YES.

The minimum-path-length between $\times$ and $\circ$ is $4$.

It can take long, but it works.
Is Optimization Really Harder than Decision?

If you can solve the decision problem, you can solve the optimization problem.

- Is there a path in the graph between nodes $\otimes$ and $\odot$ of length at most 1? **NO**
- Is there a path in the graph between nodes $\otimes$ and $\odot$ of length at most 2? **NO**
- Is there a path in the graph between nodes $\otimes$ and $\odot$ of length at most 3? **NO**
- Is there a path in the graph between nodes $\otimes$ and $\odot$ of length at most 4? **YES**

You ask the decision question until the answer is **YES**.

The minimum-pathlength between $\otimes$ and $\odot$ is 4.

It can take long, but it works.

Decision and optimization are “equivalent” when it comes to solvability.

A computing problem is a decision problem.
Standard formulation of a decision problem:
Standard formulation of a decision problem:

**Problem:** GRAPH-DISTANCE-\(D\)
Standard formulation of a decision problem:

**Problem:** GRAPH-DISTANCE-$D$

**Input:** Finite graph $G$; nodes $x$, $y$; target distance $D$. 
Standard formulation of a decision problem:

**Problem:** GRAPH-DISTANCE-\(D\)
**Input:** Finite graph \(G\); nodes \(x, y\); target distance \(D\).
**Question:** Is there an \((x,y)\)-path in \(G\) of length at most \(D\).
Standard formulation of a decision problem:

**Problem:** GRAPH-DISTANCE-$D$

**Input:** Finite graph $G$; nodes $x, y$; target distance $D$.

**Question:** Is there an $(x,y)$-path in $G$ of length at most $D$.

Every decision problem has a **YES**-set, which we usually don’t explicitly list.

**YES**-set = \{input strings $w$ for which the answer is **YES**\}

= \{w_1, w_2, w_3, \ldots \}.

← A *language* is any set of finite binary strings
Standard formulation of a decision problem:

**Problem:** \textsc{graph-distance-}D  
**Input:** Finite graph \( G \); nodes \( x, y \); target distance \( D \).  
**Question:** Is there an \((x,y)\)-path in \( G \) of length at most \( D \).

Every decision problem has a **YES**-set, which we usually don’t explicitly list.

\[
\text{**YES**-set} = \{ \text{input strings } w \text{ for which the answer is **YES**} \} = \{ w_1, w_2, w_3, \ldots \}.
\]

---

A computing problem is a **YES**-set, a set of finite binary strings.
Language: Set of finite binary strings.
Computing Problems Are Languages

**Language:** Set of finite binary strings.

**Solving the problem**
Give a “procedure” to tell if a general input $w$ is in the language (YES-set).

Abstract, precise and general formulation of a computing problem.
**Language**: Set of finite binary strings.

**Solving the problem**
Give a “procedure” to tell if a general input $w$ is in the language (YES-set).

Abstract, precise and general formulation of a computing problem.

$$\{\varepsilon, 1, 10, 01\} \quad \leftarrow \text{finite language}$$


Language: Set of finite binary strings.

Solving the problem

Give a “procedure” to tell if a general input \( w \) is in the language (YES-set).

Abstract, precise and general formulation of a computing problem.

\[
\begin{align*}
\{\epsilon, 1, 10, 01\} & \quad \leftarrow \text{finite language} \\
\Sigma^* & \quad \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\} & \quad \leftarrow \text{all finite strings}
\end{align*}
\]
Computing Problems Are Languages

Language: Set of finite binary strings.

Solving the problem
Give a “procedure” to tell if a general input \( w \) is in the language (\( \text{YES}-\)set).

Abstract, precise and general formulation of a computing problem.

\[
\begin{align*}
\Sigma^* & \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\} & \leftarrow \text{finite language} \\
L_{\text{prime}} & \{10, 11, 101, 111, 1011, 1101, 10001, \ldots\} & \leftarrow \text{all finite strings}
\end{align*}
\]
Computing Problems Are Languages

**Language:** Set of finite binary strings.

**Solving the problem**
Give a “procedure” to tell if a general input $w$ is in the language (YES-set).

Abstract, precise and general formulation of a computing problem.

\[
\begin{align*}
\Sigma^* & \leftarrow \text{finite language} \\
\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\} & \leftarrow \text{all finite strings} \\
L_{\text{prime}} & \leftarrow \{10, 11, 101, 111, 1011, 1101, 10001, \ldots\} \\
L_{\text{push}} & \leftarrow \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, \ldots\}
\end{align*}
\]
Computing Problems Are Languages

**Language:** Set of finite binary strings.

---

**Solving the problem**

Give a “procedure” to tell if a general input $w$ is in the language (YES-set).

---

Abstract, precise and general formulation of a computing problem.

- $\{\varepsilon, 1, 10, 01\}$ ← finite language
- $\Sigma^*$ ← all finite strings
- $L_{\text{prime}} \{10, 11, 101, 111, 1011, 1101, 10001, \ldots\}$
- $L_{\text{push}} \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, \ldots\}$
- $L_{\text{door}} \{1, 11, 101, 110, 111, 1011, 1101 \ldots\}$
Computing Problems Are Languages

**Language:** Set of finite binary strings.

---

**Solving the problem**

Give a “procedure” to tell if a general input $w$ is in the language (YES-set).

Abstract, precise and general formulation of a computing problem.

\[
\begin{align*}
\Sigma^* &\quad \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\} &\quad \leftarrow \text{all finite strings} \\
\mathcal{L}_{\text{prime}} &\quad \{10, 11, 101, 111, 1011, 1101, 10001, \ldots\} \\
\mathcal{L}_{\text{push}} &\quad \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, \ldots\} \\
\mathcal{L}_{\text{door}} &\quad \{1, 11, 101, 110, 111, 1011, 1101, \ldots\} \\
\mathcal{L}_{\text{unary}} &\quad \{\varepsilon, 1, 11, 111, 1111, \ldots\} = \{1^n | n \geq 0\} &\quad \leftarrow \text{strings of 1s}
\end{align*}
\]
Computing Problems Are Languages

**Language:** Set of finite binary strings.

**Solving the problem**

Give a “procedure” to tell if a general input $w$ is in the language (YES-set).

Abstract, precise and general formulation of a computing problem.

\[
\begin{align*}
\Sigma^* & = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\} \\
L_{\text{prime}} & = \{10, 11, 101, 111, 1011, 1101, 10001, \ldots\} \\
L_{\text{push}} & = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, \ldots\} \\
L_{\text{door}} & = \{1, 11, 101, 110, 111, 1011, 1101, \ldots\} \\
L_{\text{unary}} & = \{\varepsilon, 1, 11, 111, 1111, \ldots\} = \{1^n \mid n \geq 0\} \\
L_{(01)^n} & = \{\varepsilon, 01, 0101, 010101, \ldots\} = \{(01)^n \mid n \geq 0\}
\end{align*}
\]
Computing Problems Are Languages

**Language:** Set of finite binary strings.

---

**Solving the problem**

Give a “procedure” to tell if a general input $w$ is in the language (YES-set).

---

Abstract, precise and general formulation of a computing problem.

<table>
<thead>
<tr>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Sigma^*$</td>
<td>${ \varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots }$</td>
</tr>
<tr>
<td>$\mathcal{L}_{\text{prime}}$</td>
<td>${10, 11, 101, 111, 1011, 1001, 10101, \ldots }$</td>
</tr>
<tr>
<td>$\mathcal{L}_{\text{push}}$</td>
<td>${1, 01, 11, 001, 011, 101, 111, 0001, 0011, \ldots }$</td>
</tr>
<tr>
<td>$\mathcal{L}_{\text{door}}$</td>
<td>${1, 11, 101, 110, 111, 1011, 1101, \ldots }$</td>
</tr>
<tr>
<td>$\mathcal{L}_{\text{unary}}$</td>
<td>${\varepsilon, 1, 11, 111, 1111, \ldots } = {1^n \mid n \geq 0}$</td>
</tr>
<tr>
<td>$\mathcal{L}_{(01)^n}$</td>
<td>${\varepsilon, 01, 0101, 010101, \ldots } = {(01)^n \mid n \geq 0}$</td>
</tr>
<tr>
<td>$\mathcal{L}_{0^n1^n}$</td>
<td>${01, 0011, 000111, \ldots } = {0^n1^n \mid n \geq 0}$</td>
</tr>
</tbody>
</table>

$\leftarrow$ finite language

$\leftarrow$ all finite strings

$\leftarrow$ strings of 1s
Computing Problems Are Languages

**Language:** Set of finite binary strings.

---

**Solving the problem**

Give a “procedure” to tell if a general input $w$ is in the language (YES-set).

Abstract, precise and general formulation of a computing problem.

---

<table>
<thead>
<tr>
<th>$\Sigma^*$</th>
<th>${\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots}$</th>
<th>← finite language</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{L}_{\text{prime}}$</td>
<td>${10, 11, 101, 111, 1011, 1101, 10001, \ldots}$</td>
<td>← all finite strings</td>
</tr>
<tr>
<td>$\mathcal{L}_{\text{push}}$</td>
<td>${1, 01, 11, 001, 011, 101, 111, 0001, 0011, \ldots}$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{L}_{\text{door}}$</td>
<td>${1, 11, 101, 110, 111, 1011, 1101, \ldots}$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{L}_{\text{unary}}$</td>
<td>${\varepsilon, 1, 11, 111, 1111, \ldots} = {1^n \mid n \geq 0}$</td>
<td>← strings of 1s</td>
</tr>
<tr>
<td>$\mathcal{L}_{(01)^n}$</td>
<td>${\varepsilon, 01, 0101, 010101, \ldots} = {(01)^n \mid n \geq 0}$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{L}_{0^n1^n}$</td>
<td>${01, 0011, 000111, \ldots} = {0^n1^n \mid n \geq 0}$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{L}_{\text{pal}}$</td>
<td>${\varepsilon, 0, 1, 00, 11, 000, 010, 101, 111, \ldots}$</td>
<td>← palindromes</td>
</tr>
</tbody>
</table>
Computing Problems Are Languages

Language: Set of finite binary strings.

Solving the problem
Give a “procedure” to tell if a general input $w$ is in the language ($\text{YES}$-set).

Abstract, precise and general formulation of a computing problem.

\[
\begin{align*}
\{\varepsilon, 1, 10, 01\} & \quad \leftarrow \text{finite language} \\
\Sigma^* & \quad \leftarrow \text{all finite strings} \\
\mathcal{L}_\text{prime} & \quad \{10, 11, 101, 111, 1011, 1101, 10001, \ldots\} \\
\mathcal{L}_\text{push} & \quad \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, \ldots\} \\
\mathcal{L}_\text{door} & \quad \{1, 11, 101, 110, 111, 1011, 1101, \ldots\} \\
\mathcal{L}_\text{unary} & \quad \{\varepsilon, 1, 11, 111, 1111, \ldots\} = \{1^n \mid n \geq 0\} \\
\mathcal{L}_{(01)^n} & \quad \{\varepsilon, 01, 0101, 010101, \ldots\} = \{(01)^n \mid n \geq 0\} \\
\mathcal{L}_{0^n1^n} & \quad \{01, 0011, 000111, \ldots\} = \{0^n1^n \mid n \geq 0\} \\
\mathcal{L}_\text{pal} & \quad \{\varepsilon, 0, 1, 00, 11, 000, 010, 101, 111, \ldots\} \\
\mathcal{L}_\text{repeated} & \quad \{\varepsilon, 00, 11, 0000, 0101, 1010, 1111, \ldots\} \\
\end{align*}
\]
Describing a Language: String Patterns and Variables

An example where there is a clear pattern,

\[ \mathcal{L} = \{ \varepsilon, 01, 0101, 010101, \ldots \} . \]
An example where there is a clear pattern,

\[ \mathcal{L} = \{ \varepsilon, 01, 0101, 010101, \ldots \}. \]

Use a variable to formally define \( \mathcal{L} \):

\[ \mathcal{L} = \{ w \mid w = (01)^n, \text{ where } n \geq 0 \}. \quad \text{(informally } \{ (01)^n \mid n \geq 0 \} \text{)} \]
Describing a Language: String Patterns and Variables

An example where there is a clear pattern,

\[ \mathcal{L} = \{ \varepsilon, 01, 0101, 010101, \ldots \} . \]

Use a variable to formally define \( \mathcal{L} \):

\[
\mathcal{L} = \{ w \mid w = (01)^n, \text{ where } n \geq 0 \}. 
\text{ (informally } \{ (01)^n \mid n \geq 0 \} \text{ )}
\]

More than one variable:
Describing a Language: String Patterns and Variables

An example where there is a clear pattern,

\[ \mathcal{L} = \{ \varepsilon, 01, 0101, 010101, \ldots \}. \]

Use a variable to formally define \( \mathcal{L} \):

\[ \mathcal{L} = \{ w \mid w = (01)^n, \text{ where } n \geq 0 \}. \quad \text{(informally } \{ (01)^n \mid n \geq 0 \} \text{)} \]

More than one variable:

\[ \{ u \cdot v \mid u \in \Sigma^* \text{ and } v = u^R \} = \{ \varepsilon, 00, 11, 0000, 0110, 1001, 1111, \ldots \}. \quad \text{← even palindromes} \]

**Exercise.** Define \( \mathcal{L}_{\text{add}} = \{ 0100, 011000, 001000, 00110000, 00010000, 0001100000, 01100000, 0011000000, 000111000000, \ldots \} \)
Describing a Language: String Patterns and Variables

An example where there is a clear pattern,

\[ \mathcal{L} = \{ \varepsilon, 01, 0101, 010101, \ldots \}. \]

Use a variable to formally define \( \mathcal{L} \):

\[ \mathcal{L} = \{ w \mid w = (01)^n, \text{ where } n \geq 0 \}. \quad \text{(informally } \{ (01)^n \mid n \geq 0 \} \text{)} \]

More than one variable:

\[ \{ u \cdot v \mid u \in \Sigma^* \text{ and } v = u^R \} = \{ \varepsilon, 00, 11, 0000, 0110, 1001, 1111, \ldots \}. \quad \leftarrow \text{even palindromes} \]

**Exercise.** Define \( \mathcal{L}_{\text{add}} = \{ 0100, 011000, 001000, 00110000, 00010000, 0011100000, 01110000, 0011100000, 000111000000, \ldots \} \)

Ans: \( \{ 0^n \cdot 1^m \cdot 0^{n+m} \} \)

For more complicated patterns, we use regular expressions, e.g. the Unix/Linux command

```
ls FOCS*
```

(Lists everything that starts with FOCS (* is the “wild-card”).)
The Regular Expression: $\{1, 11\} \cdot \{0, 01\}^* \cdot (\{00\} \cup \{1\}^*)$

Basic building blocks are finite languages:

$\{1, 11\}$  $\{0, 01\}$  $\{00\}$  $\{1\}$
The Regular Expression: $\{1, 11\} \bullet \{0, 01\}^* \bullet (\{00\} \cup \{1\}^*)$

Basic building blocks are finite languages:

$$\{1, 11\} \quad \{0, 01\} \quad \{00\} \quad \{1\}$$

Combine these using

union, intersection, complement \hspace{1cm} (Familiar.)
The Regular Expression: $\{1, 11\} \bullet \{0, 01\}^* \bullet (\{00\} \cup \{1\}^*)$

Basic building blocks are finite languages:

$\{1, 11\}$ $\{0, 01\}$ $\{00\}$ $\{1\}$

Combine these using

- union, intersection, complement (Familiar.)
- concatenation $\bullet$, Kleene-star $^*$ (What?!?)
The Regular Expression: $\{1, 11\} \bullet \{0, 01\}^* \bullet (\{00\} \cup \{1\}^*)$

Basic building blocks are finite languages:

$\{1, 11\} \quad \{0, 01\} \quad \{00\} \quad \{1\}$

Combine these using

- union, intersection, complement (Familiar.)
- concatenation $\bullet$, Kleene-star $^*$ (What?!?)

Concatenation of languages.

$\mathcal{L}_1 \bullet \mathcal{L}_2 \bullet \mathcal{L}_3 = \{w_1 \bullet w_2 \bullet w_3 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2, w_3 \in \mathcal{L}_3\}.$

$\{0, 01\} \bullet \{0, 11\} = \{00, 011, 010, 0111\}$
The Regular Expression: \( \{1, 11\} \bullet \{0, 01\}^* \bullet (\{00\} \cup \{1\}^*) \)

Basic building blocks are finite languages:

\[
\{1, 11\} \quad \{0, 01\} \quad \{00\} \quad \{1\}
\]

Combine these using

- union, intersection, complement  
  (Familiar.)
- concatenation \(\bullet\), Kleene-star \(^*\)  
  (What?!?)

**Concatenation of languages.**

\[
\mathcal{L}_1 \bullet \mathcal{L}_2 \bullet \mathcal{L}_3 = \{w_1 \bullet w_2 \bullet w_3 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2, w_3 \in \mathcal{L}_3\}.
\]

- \(\{0, 01\} \bullet \{0, 11\} = \{00, 011, 010, 0111\}\)
- \(\{0, 11\} \bullet \{0, 01\} = \{00, 001, 110, 1101\}\)
- \(\mathcal{L}_1 \bullet \mathcal{L}_2 \neq \mathcal{L}_2 \bullet \mathcal{L}_1\)
The Regular Expression: \( \{1, 11\} \bullet \{0, 01\}^* \bullet (\{00\} \cup \{1\}^*) \)

Basic building blocks are finite languages:
\[
\{1, 11\} \quad \{0, 01\} \quad \{00\} \quad \{1\}
\]

Combine these using
- union, intersection, complement (Familiar.)
- concatenation \( \bullet \), Kleene-star \( ^* \) (What?!?)

Concatenation of languages.

\[
\mathcal{L}_1 \bullet \mathcal{L}_2 \bullet \mathcal{L}_3 = \{w_1 \bullet w_2 \bullet w_3 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2, w_3 \in \mathcal{L}_3\}.
\]

\[
\begin{align*}
\{0, 01\} \bullet \{0, 11\} &= \{00, 011, 010, 0111\} \\
\{0, 11\} \bullet \{0, 01\} &= \{00, 001, 110, 1101\} \\
\{0, 01\} \bullet \{0, 01\} &= \{0, 01\}\cdot^2 = \{00, 001, 010, 0101\}
\end{align*}
\]

\( \mathcal{L}_1 \bullet \mathcal{L}_2 \neq \mathcal{L}_2 \bullet \mathcal{L}_1 \)

(self-concatenation)

**Pop Quiz.** What is \( \{0, 01\} \bullet \{1, 10\} \)? What is \( \{0, 01\}\cdot^3 \)? What is \( \{0, 01\}\cdot^6 \)?
The Regular Expression: \( \{1, 11\} \cdot \{0, 01\}^* \cdot (\{00\} \cup \{1\}^*) \)

Basic building blocks are finite languages:

\[
\{1, 11\} \quad \{0, 01\} \quad \{00\} \quad \{1\}
\]

Combine these using

union, intersection, complement (Familiar.)
concatenation \(\cdot\), Kleene-star \(\ast\) (What?!?)

Concatenation of languages.

\[
\mathcal{L}_1 \cdot \mathcal{L}_2 \cdot \mathcal{L}_3 = \{ w_1 \cdot w_2 \cdot w_3 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2, w_3 \in \mathcal{L}_3 \}.
\]

\[
\begin{align*}
\{0, 01\} \cdot \{0, 11\} &= \{00, 011, 010, 0111\} \\
\{0, 11\} \cdot \{0, 01\} &= \{00, 001, 110, 1101\} \\
\{0, 01\} \cdot \{0, 01\} &= \{01\}^2 = \{00, 001, 010, 0101\} \\
\end{align*}
\]

\(\mathcal{L}_1 \cdot \mathcal{L}_2 \neq \mathcal{L}_2 \cdot \mathcal{L}_1\) (self-concatenation)

**Pop Quiz.** What is \(\{0, 01\} \cdot \{1, 10\}\)? What is \(\{0, 01\}^3\)? What is \(\{0, 01\}^0\)?

**Kleene star:** All possible concatenations of a finite number of strings from a language.

\[
\{0, 01\}^* = \{\varepsilon, 0, 01, 00, 011, 010, 0111, \ldots\}
\]
The Regular Expression: \( \{1, 11\} \cdot \{0, 01\}^* \cdot (\{00\} \cup \{1\}^*) \)

Basic building blocks are finite languages:
\[
\begin{align*}
\{1, 11\} & \quad \{0, 01\} & \quad \{00\} & \quad \{1\}
\end{align*}
\]

Combine these using
- union, intersection, complement (Familiar.)
- concatenation \(\cdot\), Kleene-star \(\star\) (What?!?)

**Concatenation of languages.**

\[
\mathcal{L}_1 \cdot \mathcal{L}_2 \cdot \mathcal{L}_3 = \{w_1 \cdot w_2 \cdot w_3 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2, w_3 \in \mathcal{L}_3\}.
\]

\[
\begin{align*}
\{0, 01\} \cdot \{0, 11\} &= \{00, 011, 010, 0111\} \\
\{0, 11\} \cdot \{0, 01\} &= \{00, 001, 110, 1101\} \\
\{0, 01\} \cdot \{0, 01\} &= \{0, 01\}^2 = \{00, 001, 010, 0101\} \\
\mathcal{L}_1 \cdot \mathcal{L}_2 &\neq \mathcal{L}_2 \cdot \mathcal{L}_1 \\
\{0, 01\}^2 &= \{00, 001, 010, 0101\} & \text{(self-concatenation)}
\end{align*}
\]

**Pop Quiz.** What is \(\{0, 01\} \cdot \{1, 10\}\)? What is \(\{0, 01\}^3\)? What is \(\{0, 01\}^\omega\)?

**Kleene star:** All possible concatenations of a finite number of strings from a language.

\[
\{0, 01\}^* = \{\varepsilon, 0, 01, \}
\]
The Regular Expression: \( \{1, 11\} \bullet \{0, 01\}^* \bullet (\{00\} \cup \{1\}^*) \)

Basic building blocks are finite languages:
\[
\{1, 11\} \quad \{0, 01\} \quad \{00\} \quad \{1\}
\]

Combine these using
- union, intersection, complement
- (Familiar.)
- concatenation \( \bullet \), Kleene-star \( ^* \)
- (What?!?)

**Concatenation of languages.**

\[
\mathcal{L}_1 \bullet \mathcal{L}_2 \bullet \mathcal{L}_3 = \{w_1 \bullet w_2 \bullet w_3 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2, w_3 \in \mathcal{L}_3\}.
\]

\[
\{0, 01\} \bullet \{0, 11\} = \{00, 011, 010, 0111\}
\]
\[
\{0, 11\} \bullet \{0, 01\} = \{00, 001, 110, 1101\}
\]
\[
\{0, 01\} \bullet \{0, 01\} = \{00, 01\}^2 = \{00, 001, 010, 0101\}
\]
\[
\mathcal{L}_1 \bullet \mathcal{L}_2 \neq \mathcal{L}_2 \bullet \mathcal{L}_1
\]

(self-concatenation)

**Pop Quiz.** What is \( \{0, 01\} \bullet \{1, 10\} \)? What is \( \{0, 01\}^3 \)? What is \( \{0, 01\}^6 \)?

**Kleene star:** All possible concatenations of a finite number of strings from a language.
\[
\{0, 01\}^* = \{\varepsilon, 0, 01, 00, 001, 010, 0101, \}
\]
The Regular Expression: $\{1, 11\} \bullet \{0, 01\}^* \bullet (\{00\} \cup \{1\}^*)$

Basic building blocks are finite languages:

$$\{1, 11\} \quad \{0, 01\} \quad \{00\} \quad \{1\}$$

Combine these using

- union, intersection, complement (Familiar.)
- concatenation $\bullet$, Kleene-star $^*$ (What?!!?)

**Concatenation of languages.**

$$\mathcal{L}_1 \bullet \mathcal{L}_2 \bullet \mathcal{L}_3 = \{ w_1 \bullet w_2 \bullet w_3 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2, w_3 \in \mathcal{L}_3 \}.$$  

$$\{0, 01\} \bullet \{0, 11\} = \{00, 011, 010, 0111\}$$
$$\{0, 11\} \bullet \{0, 01\} = \{00, 001, 110, 1101\}$$
$$\{0, 01\} \bullet \{0, 01\} = \{0, 01\}^2 = \{00, 001, 010, 0101\}$$

$$\mathcal{L}_1 \bullet \mathcal{L}_2 \neq \mathcal{L}_2 \bullet \mathcal{L}_1$$ (self-concatenation)

**Pop Quiz.** What is $\{0, 01\} \bullet \{1, 10\}$? What is $\{0, 01\}^3$? What is $\{0, 01\}^6$?

**Kleene star:** All possible concatenations of a finite number of strings from a language.

$$\{0, 01\}^* = \{\varepsilon, 0, 01, 00, 001, 010, 0101, 000, 0001, 0010, \ldots\} = \bigcup_{n=0}^{\infty} \{0, 01\}^n.$$
The Regular Expression: \( \{1, 11\} \bullet \{0, 01\}^* \bullet (\{00\} \cup \{1\}^*) \)

Basic building blocks are finite languages:

\[ \{1, 11\}, \{0, 01\}, \{00\}, \{1\} \]

Combine these using union, intersection, complement (Familiar.)

concatenation \(\bullet\), Kleene-star * (What?!?)

### Concatenation of languages.

\[ \mathcal{L}_1 \bullet \mathcal{L}_2 \bullet \mathcal{L}_3 = \{ w_1 \bullet w_2 \bullet w_3 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2, w_3 \in \mathcal{L}_3 \} \]

\[ \{0, 01\} \bullet \{0, 11\} = \{00, 011, 010, 0111\} \]
\[ \{0, 11\} \bullet \{0, 01\} = \{00, 001, 110, 1101\} \]
\[ \{0, 01\} \bullet \{0, 01\} = \{0, 01\}^2 = \{00, 001, 010, 0101\} \]
\[ \mathcal{L}_1 \bullet \mathcal{L}_2 \neq \mathcal{L}_2 \bullet \mathcal{L}_1 \]

**self-concatenation**

### Pop Quiz. What is \(\{0, 01\} \bullet \{1, 10\}\)? What is \(\{0, 01\}^3\)? What is \(\{0, 01\}^0\)?

### Kleene star: All possible concatenations of a finite number of strings from a language.

\[ \{0, 01\}^* = \{\varepsilon, 0, 01, 00, 001, 010, 0101, 000, 0001, 0010, \ldots\} \]
\[ = \bigcup_{n=0}^{\infty} \{0, 01\}^n; \]

\[ \{1\}^* = \{\varepsilon, 1, 11, 111, 1111, 11111, \ldots\} \]
\[ = \bigcup_{n=0}^{\infty} \{1\}^n. \]

**Pop Quiz.** Which of the strings \(\{101110, 00111, 00100, 01100\}\) can you generate using \(\{0, 01\}^* \bullet \{1, 10\}^*\)?
The Regular Expression: \( \{1, 11\} \bullet \overline{\{0, 01\}}^\ast \bullet (\{00\} \cup \{1\}^\ast) \)

\[
\begin{align*}
\{0, 01\}^\ast &= \{\varepsilon, 0, 01, 00, 001, 010, 0101, 000, 0010, \ldots\} \\
\{1\}^\ast &= \{\varepsilon, 1, 11, 111, 1111, 11111, \ldots\}
\end{align*}
\]

To generate 1110111:

\[11 \in \{1, 11\}\]
The Regular Expression: $\{1, 11\} \bullet \{0, 01\}^* \bullet (\{00\} \cup \{1\}^*)$

\[
\{0, 01\}^* = \{\varepsilon, 0, 01, 00, 001, 010, 0101, 000, 0010, \ldots\}
\]
\[
\{1\}^* = \{\varepsilon, 1, 11, 111, 1111, 11111, \ldots\}
\]

To generate 110111:

\[
11 \in \{1, 11\}
\]
\[
10 \in \{0, 01\}^*
\]
The Regular Expression: \( \{1, 11\} \bullet \overline{0, 01}^* \bullet (\{00\} \cup \{1\}^*) \)

\[
\{0, 01\}^* = \{\varepsilon, 0, 01, 00, 001, 010, 0101, 000, 0010, \ldots\} \\
\{1\}^* = \{\varepsilon, 1, 11, 111, 1111, 11111, \ldots\}
\]

To generate 1110111:

\[
11 \in \{1, 11\} \\
10 \in \overline{0, 01}^* \\
111 \in \{00\} \cup \{1\}^*
\]

Hence 1110111 \( \in \{1, 11\} \bullet \overline{0, 01}^* \bullet (\{00\} \cup \{1\}^*) \)

**Pop Quiz** Is there another way to generate 1110111?

**Pop Quiz** Yes or no: 11110010 \( \in \{1, 11\} \bullet \overline{0, 01}^* \bullet (\{00\} \cup \{1\}^*) \)?
Is there a simple procedure to test if a given string satisfies a regular expression?
Is there a simple procedure to test if a given string satisfies a regular expression?

\[11110010 \in \{1, 11\} \cdot \overline{0, 01}^* \cdot (\{00\} \cup \{1\}^*) \quad ???\]
Challenges Involving Regular Expressions

1. Is there a simple procedure to test if a given string satisfies a regular expression?

\[11100010 \in \{1, 11\} \cdot \overline{\{0, 01\}}^* \cdot (\{00\} \cup \{1\}^*)\] ???

2. Regular expression for all palindromes (strings which equal their reversal)?
Recursively Defined Languages: Palindromes

1. \( \varepsilon, 0, 1 \in \mathcal{L}_{\text{palindrome}} \).

[basis]
Recursively Defined Languages: Palindromes

1. \( \varepsilon, 0, 1 \in \mathcal{L}_{\text{palindrome}} \).

2. \( w \in \mathcal{L}_{\text{palindrome}} \rightarrow 0 \cdot w \cdot 0 \in \mathcal{L}_{\text{palindrome}}, \)
   \( 1 \cdot w \cdot 1 \in \mathcal{L}_{\text{palindrome}} \).

[basis]

[constructor rules]
Recursively Defined Languages: Palindromes

1. $\varepsilon, 0, 1 \in \mathcal{L}_{\text{palindrome}}$. [basis]

2. $w \in \mathcal{L}_{\text{palindrome}} \rightarrow 0 \cdot w \cdot 0 \in \mathcal{L}_{\text{palindrome}},$
   $1 \cdot w \cdot 1 \in \mathcal{L}_{\text{palindrome}}$. [constructor rules]

3. Nothing else is in $\mathcal{L}_{\text{palindrome}}$. [minimality]
Recursively Defined Languages: Palindromes

1. $\varepsilon, 0, 1 \in \mathcal{L}_{\text{palindrome}}$. 

2. $w \in \mathcal{L}_{\text{palindrome}} \rightarrow 0 \cdot w \cdot 0 \in \mathcal{L}_{\text{palindrome}},$
   $1 \cdot w \cdot 1 \in \mathcal{L}_{\text{palindrome}}$. 

3. Nothing else is in $\mathcal{L}_{\text{palindrome}}$. 

[ basis ]

[ constructor rules ]

[ minimality ]

Pop Quiz. Similar looking languages: $\{0^n1^k \mid n, k \geq 0\}$ and $\{0^n1^n \mid n \geq 0\}$
Recursively Defined Languages: Palindromes

1. \( \varepsilon, 0, 1 \in L_{\text{palindrome}} \).

2. \( w \in L_{\text{palindrome}} \rightarrow 0 \cdot w \cdot 0 \in L_{\text{palindrome}} \),
   \( 1 \cdot w \cdot 1 \in L_{\text{palindrome}} \).

3. Nothing else is in \( L_{\text{palindrome}} \).

[basis]  [constructor rules]  [minimality]

Pop Quiz. Similar looking languages: \( \{0^n1^k \mid n, k \geq 0\} \)  and  \( \{0^n1^n \mid n \geq 0\} \)
Give recursive definitions of these languages.
Recursively Defined Languages: Palindromes

1. $\varepsilon, 0, 1 \in \mathcal{L}_{\text{palindrome}}$.  
   [basis]

2. $w \in \mathcal{L}_{\text{palindrome}} \rightarrow 0\cdot w \cdot 0 \in \mathcal{L}_{\text{palindrome}}$,  
   $1\cdot w \cdot 1 \in \mathcal{L}_{\text{palindrome}}$.  
   [constructor rules]

3. Nothing else is in $\mathcal{L}_{\text{palindrome}}$.  
   [minimality]

**Pop Quiz.** Similar looking languages: $\{0^n1^k \mid n, k \geq 0\}$ and $\{0^n1^n \mid n \geq 0\}$

Give recursive definitions of these languages.

Give regular expressions for these languages.
Recursively Defined Languages: Palindromes

1. \( \varepsilon, 0, 1 \in \mathcal{L}_{\text{palindrome}}. \) [basis]

2. \( w \in \mathcal{L}_{\text{palindrome}} \rightarrow 0 \cdot w \cdot 0 \in \mathcal{L}_{\text{palindrome}}, \)
   \( 1 \cdot w \cdot 1 \in \mathcal{L}_{\text{palindrome}}. \) [constructor rules]

3. Nothing else is in \( \mathcal{L}_{\text{palindrome}}. \) [minimality]

Pop Quiz. Similar looking languages: \( \{0^n1^k \mid n, k \geq 0\} \) and \( \{0^n1^n \mid n \geq 0\} \)

Give recursive definitions of these languages.
Give regular expressions for these languages.

These computing problems look similar.
Recursively Defined Languages: Palindromes

1. \( \varepsilon, 0, 1 \in L_{\text{palindrome}} \).  
   
   [basis]

2. \( w \in L_{\text{palindrome}} \rightarrow 0 \cdot w \cdot 0 \in L_{\text{palindrome}}, \)  
   \( 1 \cdot w \cdot 1 \in L_{\text{palindrome}} \).  
   
   [constructor rules]

3. Nothing else is in \( L_{\text{palindrome}} \).  
   
   [minimality]

**Pop Quiz.** Similar looking languages: \( \{0^n 1^k | n, k \geq 0\} \) and \( \{0^n 1 \cdot n | n \geq 0\} \)

Give recursive definitions of these languages.
Give regular expressions for these languages.

These computing problems look similar.

They are **VERY** different. Which do you think is more “complex”??
Recursively Defined Languages: Palindromes

1. $\varepsilon, 0, 1 \in L_{\text{palindrome}}$. [basis]
2. $w \in L_{\text{palindrome}} \rightarrow 0 \cdot w \cdot 0 \in L_{\text{palindrome}}$, $1 \cdot w \cdot 1 \in L_{\text{palindrome}}$. [constructor rules]
3. Nothing else is in $L_{\text{palindrome}}$. [minimality]

**Pop Quiz.** Similar looking languages: $\{0^n1^k \mid n, k \geq 0\}$ and $\{0^n1^n \mid n \geq 0\}$

Give recursive definitions of these languages.
Give regular expressions for these languages.

These computing problems look similar.

They are **VER Y** different. Which do you think is more “complex”?

How to define complexity of a computing problem?
Complexity of a Computing Problem
\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \quad \text{(strings ending in 1)} \]
Complexity of a Computing Problem

\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \]  
(strings ending in 1)

difficult problem
Complexity of a Computing Problem

\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \]  (strings ending in 1)

difficult problem \iff “complex” \text{ YES-set}
\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \] (strings ending in 1)

difficult problem \iff \text{“complex” (YES)-set} \iff \text{hard to test membership in (YES)-set}
Complexity of a Computing Problem

\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \]  
(strings ending in 1)

difficult problem \iff \text{“complex” (YES)-set} \iff \text{hard to test membership in (YES)-set}

How do we test membership?
Complexity of a Computing Problem

\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \]  
(strings ending in 1)

Difficult problem \iff \text{“complex” YES}-set \iff \text{hard to test membership in YES}-set

How do we test membership? That brings us to \textit{Models Of Computing}. 
Complexity of a Computing Problem

\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \quad \text{(strings ending in 1)} \]

difficult problem \leftrightarrow \text{“complex” YES-set} \leftrightarrow \text{hard to test membership in YES-set}

How do we test membership? That brings us to *Models Of Computing.*

\[ 1 \ 1 \ 0 \ 1 \]
Complexity of a Computing Problem

\[ L_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \]  

(strings ending in 1)

difficult problem \iff \text{“complex” YES-set} \iff \text{hard to test membership in YES-set}

How do we test membership? That brings us to Models Of Computing.

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 \\
\triangle & \rightarrow & q_0 & q_1
\end{array}
\]
Complexity of a Computing Problem

\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \] (strings ending in 1)

difficult problem \iff \text{“complex” (YES)-set} \iff \text{hard to test membership in (YES)-set}

How do we test membership? That brings us to \textit{Models Of Computing}.

Visual encoding of four (machine-level) instructions:
Complexity of a Computing Problem

\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \]  
(strings ending in 1)

difficult problem \( \iff \) “complex” \( \text{YES}\)-set \( \iff \) hard to test membership in \( \text{YES}\)-set

How do we test membership? That brings us to *Models Of Computing*.

Visual encoding of four (machine-level) instructions:

1. In state \( q_0 \), when you process a 0, transition to state \( q_0 \).
Complexity of a Computing Problem

\[ \mathcal{L}_{\text{push}} = \{ 1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \quad \text{(strings ending in 1)} \]

difficult problem \iff “complex” \text{ (YES)-set} \iff hard to test membership in \text{ (YES)-set}

How do we test membership? That brings us to *Models Of Computing.*

Visual encoding of four (machine-level) instructions:

1. In state \( q_0 \), when you process a 0, transition to state \( q_0 \).
2. In state \( q_0 \), when you process a 1, transition to state \( q_1 \).
Complexity of a Computing Problem

$\mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots\}$ (strings ending in 1)

difficult problem $\leftrightarrow$ “complex” \text{YES}-set $\leftrightarrow$ hard to test membership in \text{YES}-set

How do we test membership? That brings us to \textit{Models Of Computing}.

Visual encoding of four (machine-level) instructions:

1: In state $q_0$, when you process a 0, transition to state $q_0$.
2: In state $q_0$, when you process a 1, transition to state $q_1$.
3: In state $q_1$, when you process a 0, transition to state $q_0$. 
Complexity of a Computing Problem

\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \quad \text{(strings ending in 1)} \]

difficult problem \iff \text{“complex” YES-set} \iff \text{hard to test membership in YES-set}

How do we test membership? That brings us to *Models Of Computing*.

Visual encoding of four (machine-level) instructions:

1. In state \( q_0 \), when you process a 0, transition to state \( q_0 \).
2. In state \( q_0 \), when you process a 1, transition to state \( q_1 \).
3. In state \( q_1 \), when you process a 0, transition to state \( q_0 \).
4. In state \( q_1 \), when you process a 1, transition to state \( q_1 \).
Complexity of a Computing Problem

\[ \mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \ldots \} \]  
(strings ending in 1)

difficult problem \quad \leftrightarrow \quad \text{“complex” (YES)-set} \quad \leftrightarrow \quad \text{hard to test membership in (YES)-set}

How do we test membership? That brings us to \textit{Models Of Computing}.

Visual encoding of four (machine-level) instructions:

1: In state \( q_0 \), when you process a 0, transition to state \( q_0 \).
2: In state \( q_0 \), when you process a 1, transition to state \( q_1 \).
3: In state \( q_1 \), when you process a 0, transition to state \( q_0 \).
4: In state \( q_1 \), when you process a 1, transition to state \( q_1 \).

“Easy” to implement as a mechanical device.
A Simple Computing Machine (DFA)

$q_0$ (current state in gray)
A Simple Computing Machine (DFA)

(q₀, q₁)

0 1 1

(current state in gray)
A Simple Computing Machine (DFA)

1 1 0 1

(q current state in gray)

1 1 0 1
A Simple Computing Machine (DFA)

1 1 0 1

q₀

0

q₁

1

1

0

(1)

(2)

(3)

(4)

1 1 0 1

&

(current state in gray)
A Simple Computing Machine (DFA)

1 1 0 1
\[ \Delta \]

(q0) 0 \rightarrow 1 \rightarrow 1

1 1 0 1
\[ \Delta \]

(q0)(q1)
0 \rightarrow 1 \rightarrow 1

1 1 0 1
\[ \Delta \]

(q0)(q1)
0 \rightarrow 1 \rightarrow 1

(current state in gray)
A Simple Computing Machine (DFA)

1101

q0 1 1

q1

0 1

(1 current state in gray)

1101

q0 1 1

q1

0 1

1101

q0 1 1

q1

0 1

1101

q0 1 1

q1

0 1

(current state in gray)
A Simple Computing Machine (DFA)

(current state in gray)
A Simple Computing Machine (DFA)

1 1 0 1

(current state in gray)
A Simple Computing Machine (DFA)

1 1 0 1
\[ \triangleleft \]

Current state in gray

1 1 0 1
\[ \triangleleft \]

1 1 0 1
\[ \triangleleft \]

1 1 0 1
\[ \triangleleft \]

1 1 0 1
\[ \triangleleft \]

1 1 0 1
\[ \triangleleft \]
A Simple Computing Machine (DFA)

\[ L_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, \ldots\} \]

Strings in \( L_{\text{push}} \) end in the "accepting" state \( q_1 \).
Strings not in \( L_{\text{push}} \) do not.
Computing Problems and Their Difficulty

- Computing Problem
- Decision Problem
Computing Problems and Their Difficulty

Language $\mathcal{L}$: \text{(YES)-set of finite binary strings}
Computing Problems and Their Difficulty

How hard is the problem?

Language $\mathcal{L}$: yes-set of finite binary strings
Computing Problems and Their Difficulty

- **Computing Problem**
- **Decision Problem**
  - How hard is the problem?
  - How complex is $\mathcal{L}$?
  - How hard is it to test membership in $\mathcal{L}$?

Language $\mathcal{L}$: (YES)-set of finite binary strings
A problem can be harder in two ways.

1. The problem needs more resources. For example, the problem can be solved with a similar machine to ours, except with more states.
A problem can be harder in two ways.

1. The problem needs more resources. For example, the problem can be solved with a similar machine to ours, except with more states.

2. The problem needs a different *kind* of computing machine, with superior capabilities.
A problem can be harder in two ways.

1. The problem needs more resources. For example, the problem can be solved with a similar machine to ours, except with more states.

2. The problem needs a different kind of computing machine, with superior capabilities.

The first type of “harder” is the focus of a follow-on algorithms course.

We focus on what can and can’t be solved on a particular kind of machine.