Languages: What is Computation?

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.
- \( \mathbb{N}_0, \mathbb{E}, \mathbb{Z}, \mathbb{Q} \) are countable.
- Finite binary strings \( \mathcal{B} \) 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.
Today: Languages: What is Computation?

1. Decision problems.

2. Languages.
   - Describing a language.

What is a Computing Problem?

\textit{Decide} \begin{itemize}
  \item [YES] or \begin{itemize}
  \item [NO]
\end{itemize}
\end{itemize} 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.
\begin{enumerate}
  \item If you come to \( n \) output \begin{itemize}
  \item [YES]
\end{itemize}.
  \item If you come to a number bigger than \( n \), output \begin{itemize}
  \item [NO]
\end{itemize}.
\end{enumerate}

Not the smartest approach to primality testing, but gets to the heart of computing
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.

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

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

Decision problems can be formulated as testing membership in a set of strings.
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

```
00110001001110010101010010000101000111100001011000110100011001100100010100100101100001100100010110000110010001011000011010000101001001010000011010000101100001100010010110000110011
```

\[ \mathcal{L}_{\text{path}} = \{ \text{All strings of the form “nodes | edges | endpoints of path | target distance” for which} \}
\]

\[ \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?

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

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

You ask the decision question until the answer is [YES].

The minimum-pathlength between $\otimes$ and $\bigcirc$ 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.
Languages

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 \texttt{YES}-set, which we usually don’t explicitly list.

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

A computing problem is a \texttt{YES}-set, a set of \textit{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.

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

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

More than one variable:

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

**Pop Quiz.** Formally define \( \mathcal{L}_{\text{add}} = \{ 0100, 011000, 001000, 00110000, 00010000, 0001100000, 01100000, 0011100000, 000111000000, \ldots \} \)

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

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

Basic building blocks are finite languages:

\[
\begin{array}{cccc}
\{1, 11\} & \{0, 01\} & \{00\} & \{1\}
\end{array}
\]

Combine these using

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

**Concatenation of languages.**

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

- \( \{0, 01\} \cdot \{0, 11\} = \{00, 011, 010, 0111\} \)
- \( \{0, 11\} \cdot \{0, 01\} = \{00, 001, 110, 1101\} \)
- \( \{0, 01\} \cdot \{0, 01\} = \{0, 01\}^* = \{00, 001, 010, 0101\} \)
- \( \mathcal{L}_1 \cdot \mathcal{L}_2 \neq \mathcal{L}_2 \cdot \mathcal{L}_1 \) (self-concatenation)

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

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

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

**Pop Quiz.** Which of the strings \( \{101110, 00111, 00100, 01100\} \) can you generate using \( \{0, 01\}^* \cdot \{1, 10\}^* \)?
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 1110111:

$11 \in \{1, 11\}$

$10 \in \{0, 01\}^*$

$111 \in \{00\} \cup \{1\}^*$

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

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

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

11110010 ∈ {1, 11} \bullet \overline{\{0, 01\}}^* \bullet (\{00\} \cup \{1\}^*)

Regular expression for all palindromes (strings which equal their reversal)?
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: $\{0^n 1^k \mid n, k \geq 0\}$ and $\{0^n 1^n \mid n \geq 0\}$

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"?

How to define complexity of a computing problem?
Complexity of a Computing Problem

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

A 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 \(q_0\).
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)

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

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

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.