Representing information

Can encode/represent any kind of data (numbers, text, pairs of numbers, graphs, images, etc…) with a finite length (binary) string over some alphabet

$$\Sigma = \{0, 1\}$$  alphabet
\[\downarrow\]  symbols of the alphabet

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

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

string of length 0 (empty string)

What is a computational problem?

Definition: A computational problem is a function

$$f : \Sigma^* \to \Sigma^*.$$

Definition: A decision problem is a function

$$f : \Sigma^* \to \{0, 1\}.$$  No, Yes
False, True
Reject, Accept

Definition: A subset $$L \subseteq \Sigma^*$$ is called a (formal) language.
There is a one-to-one correspondence between decision problems and languages.

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

\[ L = \{ w \in \Sigma^* : f(w) = 1 \} \]

Our focus will be on languages!
(decision problems)

- Convenient restriction.
- Usually “without loss of generality”.

Today

Deterministic Finite Automata (DFA)

- restricted model of computation
- very limited memory
  - reads input from left to right, and accepts or rejects.
    (one pass through the input)

State diagram of a DFA

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

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

**Input:** 1010

**Decision:** Reject

Simulation of a DFA

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

**Input:** 01111

Simulation of a DFA

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

**Input:** 01111

Simulation of a DFA

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

**Input:** 01111
Simulation of a DFA

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

**Input:** 01111

Simulation of a DFA

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

**Input:** 01111
Definition: Language decided by a DFA

Let $M$ be a DFA. We let $L(M)$ denote the set of strings that $M$ accepts.

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

If $L = L(M)$, we say that $M$ recognizes $L$.

$L(M) = \{x \in \{0, 1\}^* : x \text{ has an even number of 1's}\}$
DFA Examples

$L(M) = \text{all binary strings with even length}$
$= \{x \in \{0, 1\}^* : |x| \text{ is even}\}$

DFA Examples

$L(M) = \{x \in \{0, 1\}^* : x \text{ ends with a 0} \} \cup \{\epsilon\}$

DFA Examples

$L(M) = \{a, b, cb, cc\}$

Poll

The set of all words that contain at least two 0's
The set of all words that contain 00 as a substring
The set of all words that contain 000 as a substring
The set of all words ending in 000
The set of all words ending in 00
None of the above
Beats me
DFA construction practice

$L = \{110, 101\}$

$L = \{0, 1\}^* \setminus \{110, 101\}$

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

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

$L = \{\epsilon, 110, 110110, 110110110, \ldots\}$

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

$L = \{x \in \{0, 1\}^* : 10 \text{ and } 01 \text{ occur equally often in } x.\}$

DFA as a programming language

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

DFA as a programming language

```python
def foo(input):
    i = 0;
    STATE 0:
        if (i == input.length): return False;
        letter = input[i];
        i++;
        switch(letter):
            case '0':  go to STATE 0;
            case '1':  go to STATE 1;
    STATE 1:
        if (i == input.length): return True;
        letter = input[i];
        i++;
        switch(letter):
            case '0':  go to STATE 2;
            case '1':  go to STATE 2;
    …
```
DFA as a programming language

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

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

    ... Depending on the letter change the state.
```

Depending on the letter change the state.

Formal definition: DFA

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

where

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

---

Formal definition: DFA

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

where

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- $\delta : Q \times \Sigma \to Q$
- $q_0$ is the start state
- $F = \{q_1, q_2\}$

![DFA diagram]

![DFA diagram]
**Formal definition: DFA accepting a string**

Let \( w = w_1 w_2 \cdots w_n \) be a string over an alphabet \( \Sigma \).

Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a DFA.

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

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

Otherwise we say \( M \) **rejects** the string \( w \).

**Definition: Regular languages**

**Definition:** A language \( L \) is called **regular** if \( L = L(M) \) for some DFA \( M \).

**Regular languages**

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

- \( \{0,1\}^* \) \( \cap \) \( \{110,101\} \)
- \( \{x \in \{0,1\}^* : x \text{ starts and ends with same bit.}\} \)
- \( \{x \in \{0,1\}^* : |x| \text{ is divisible by 2 or 3}\} \)
- \( \{x, 110, 1010, 1101010, \ldots\} \)
- \( \{x \in \{0,1\}^* : x \text{ contains the substring 110.}\} \)
- \( \{x \in \{0,1\}^* : 01 \text{ and 01 occur equally often in } x.\} \)
- \( \{x \in \{0,1\}^* : \text{10 and 01 occur equally often in } x.\} \)
- \( \ldots \)
Regular languages

Questions:

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

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

A non-regular language

Theorem:

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

Note \( L = \{\varepsilon, 01, 0011, 000111, 00001111, \ldots\} \).

A non-regular language

Theorem:

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

Intuition:

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

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

Careful though:
\( L = \{x \in \{0,1\}^* : 10 \text{ and } 01 \text{ occur equally often in } x.\} \) is regular!

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

A key component of the proof:

Pigeonhole principle (PHP)
A non-regular language

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

**Input:** 0000000011111111

Imagine some arbitrary transitions

$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5$

A non-regular language

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

**Input:** 0000000011111111

Imagine some arbitrary transitions

$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5$

A non-regular language

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

**Input:** 0000000011111111

Imagine some arbitrary transitions

$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5$

A non-regular language

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

**Input:** 0000000011111111

Imagine some arbitrary transitions

$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5$
A non-regular language

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

**Input:** 0000000011111111

imagine some arbitrary transitions

00

$q_3$  $q_4$  $q_5$
A non-regular language

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

**Input:** 0000000011111111

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

But 0011 and 00000111 end up in the same state.

Pigeonhole Principle

Where will 000000 go?

A non-regular language

**Theorem:**

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

**Proof:** Proof is by contradiction. So suppose $L$ is regular.

This means there is a DFA $M$ that decides $L$.

Let $k$ denote the number of states of $M$.

Let $r_n$ denote the state $M$ is in after reading $0^n$.

By PHP, there exists $i, j \in \{0, 1, \ldots, k\}, i \neq j$, such that $r_i = r_j$. So $0^i$ and $0^j$ end up in the same state.

For any string $w$, $0^i w$ and $0^j w$ end up in the same state.

But for $w = 0^i$, $0^i w$ should end up in an accepting state, and $0^j w$ should end up in a rejecting state.

This is the desired contradiction.

Proving a language is not regular

**What makes the proof work:**

1. **Setup a proof by contradiction:**

   Assume that the language is regular.
   So a DFA with $k$ states recognizes it.

2. **Pick your pigeons:**

   Identify a set of at least $k+1$ strings as your pigeons.
   Some two of the pigeons, $x$ and $y$, must end up in the same state.
   (So for every string $z$, $x z$ and $y z$ must end up in the same state.)

3. **Reach a contradiction:**

   Find a string $z$ such that $xz \in L$ but $yz \notin L$. 
Proving a language L is not regular

**Game between adversary and you:**
Adversary:
- Picks k, the # of states in a purported DFA deciding L.
You:
- Pick a set S of k+1 distinct strings
Adversary:
- Picks two strings x and y from S
You:
- Find a string z such that exactly one of xz and yz belongs to L

Note: You don’t control the choice of k, or the pair \{x,y\} from your chosen set.
The choice of S is up to you, and can include arbitrary strings (need not be in L, need not be a nice sequence like 0^n)

To prove L is non-regular, suffices to exhibit an infinite distinguishable set S.

In fact, this characterizes non-regularity (though we won’t prove this)

Proving a language is not regular

Fix a language L.

- Call strings x,y *distinguishable* if there exists z s.t. exactly one of xz, yz belongs to L
- Call a subset S of strings *distinguishable* if every distinct pair x,y ∈ S is distinguishable

Exercise (test your understanding):

\[ \Sigma = \{a, b, c\} \]

Show that the following language is not regular:

\[ L = \{ c^{251} a^n b^{2n} : n \in \mathbb{N} \} \]

Another non-regular language?

**Question:** Are all unary languages regular?
(a language \( L \) is unary if \( L \subseteq \Sigma^* \), where \( |\Sigma| = 1 \).)

**Theorem:**
The language \( \{ 0^{2^n} : n \in \mathbb{N} \} \) is not regular.
Regular languages

Questions:

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

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

Closure properties of regular languages
**Closed under complementation**

**Proposition:**
Let \( \Sigma \) be some alphabet.
If \( L \subseteq \Sigma^* \) is regular, then so is \( \overline{L} = \Sigma^* \setminus L \).

**Proof:** If \( L \) is regular, then there is a DFA
\[
M = (Q, \Sigma, \delta, q_0, F)
\]
recognizing \( L \). Then
\[
M' = (Q, \Sigma, \delta, q_0, Q \setminus F)
\]
recognizes \( \overline{L} \). So \( \overline{L} \) is regular.

**Closed under union**

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

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

**The mindset**

**Imagine yourself as a DFA.**

**Rules:**
1) Can only scan the input once, from left to right.
2) Can only remember “constant” amount of information.

should not change based on input length

**Example**

\( L_1 = \) strings with even number of 1’s
\( L_2 = \) strings with length divisible by 3.

\[
M_1 = \begin{array}{c}
q_{even} \\
0 \\
1 \\
1 \\
0
\end{array}
\]

\[
M_2 = \begin{array}{c}
p_0 \\
0, 1 \\
p_1 \\
0, 1 \\
p_2
\end{array}
\]
Main idea: Construct a DFA that keeps track of both at once.
Main idea: Construct a DFA that keeps track of both at once.

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

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

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

Closed under union

\[
\begin{align*}
q_{\text{even } p_0} & \quad 0 \quad \rightarrow \quad q_{\text{even } p_1} \\
q_{\text{odd } p_0} & \quad 1 \quad \rightarrow \quad q_{\text{odd } p_1} \\
q_{\text{even } p_1} & \quad 0 \quad \rightarrow \quad q_{\text{odd } p_2} \\
q_{\text{odd } p_1} & \quad 1 \quad \rightarrow \quad ? \\
q_{\text{even } p_2} & \quad 0 \quad \rightarrow \quad q_{\text{odd } p_2}
\end{align*}
\]

Input: 101001
Closed under union

Input: 101001

Decision: Accept

Input: 101001

Closure under union: formal DFA definition

Proof: Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a DFA deciding \( L_1 \) and \( M' = (Q', \Sigma, \delta', q'_0, F') \) be a DFA deciding \( L_2 \). We construct a DFA \( M'' = (Q'', \Sigma, \delta'', q''_0, F'') \) that decides \( L_1 \cup L_2 \), as follows:
- \( Q'' = Q \times Q' = \{(q, q') : q \in Q, q' \in Q'\} \)
- \( \delta''((q, q'), a) = (\delta(q, a), \delta'(q', a)) \)
- \( q''_0 = (q_0, q'_0) \)
- \( F'' = \{(q, q') : q \in F \text{ or } q' \in F'\} \)

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

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

**Proof:** Follows from:
- \( L_1 \cap L_2 = \overline{L_1} \cup \overline{L_2} \)
- regular languages are closed under complementation
- regular languages are closed under union

---

Closed under concatenation (more tricky)

\[ L_1L_2 = \{xy \mid x \in L_1, y \in L_2\} \]

**Theorem:**
Let \( \Sigma \) be some alphabet. If \( L_1 \subseteq \Sigma^* \) and \( L_2 \subseteq \Sigma^* \) are regular, then so is \( L_1L_2 \).

Similarly, if \( L \subseteq \Sigma^* \) is regular, then so is \( L^* \)

---

Closed under intersection

Closure properties can be used to show languages are not regular.

**Example:**
Let \( L \subseteq \{0,1\}^* \) be the language consisting of all words with an equal number of 0’s and 1’s.

We claim \( L \) is not regular. Suppose it was regular.

\[ \{0^n1^m : n, m \in \mathbb{N}\} \cap L = \{0^n1^n : n \in \mathbb{N}\} \]

regular regular regular contradiction

---

The mindset

**Imagine yourself as a DFA.**

**Rules:**
1) Can only scan the input once, from left to right.
2) Can only remember “constant” amount of information. should not change based on input length
Given \( w \in \Sigma^* \), we need to decide if \( w = uv \) for \( u \in L_1, v \in L_2 \).

**Problem:** Don’t know where \( u \) ends, \( v \) begins.
When do you stop simulating \( M_1 \) and start simulating \( M_2 \)?

Suppose God tells you \( u \) ends at \( w_3 \).

\[
\begin{array}{ccccccccccc}
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & w_8 & w_9 & w_{10} \\
q_0 & q_1 & q_1 & q_3 & q_2 & q_1 & q_1 & q_1 & q_0 & q_0 \\
\end{array}
\]

**thread:** a simulation of \( M_1 \) and then \( M_2 \) that corresponds to breaking up input \( w \) as \( uv \) where \( u \in L_1 \).
Define a class of languages inductively as follows:

- $\emptyset$ is regular.
- For every $a \in \Sigma$, \{a\} is regular.
- $L_1, L_2$ regular $\implies L_1 \cup L_2$ regular.
- $L_1, L_2$ regular $\implies L_1 \cdot L_2$ regular.
- $L$ regular $\implies L^*$ regular.

**Theorem:** Above precisely defines regular languages

**Note:** we saw one direction, that any language thus defined is regular. Other direction skipped.

**Regular languages, recursively**

Let's define the DFA formally

\[ M_1 = (Q, \Sigma, \delta, q_0, F) \quad \quad M_2 = (Q', \Sigma, \delta', q'_0, F') \]

\[ Q'' = Q \times \mathcal{P}(Q') \]

\[ \delta'' : Q \times \mathcal{P}(Q') \times \Sigma \to Q \times \mathcal{P}(Q') \]

for $q \in Q$, $S \in \mathcal{P}(Q')$, $a \in \Sigma$:

\[ (q, S, a) \mapsto (\delta(q, a), \{\delta'(s, a) : s \in S\}) \quad \text{if} \quad \delta(q, a) \notin F \]

\[ q'' = (q_0, \emptyset) \quad \text{if} \quad q_0 \notin F \]

\[ q'' = (q_0, \{q'_0\}) \quad \text{otherwise} \]

\[ F'' = \{(q, S) : q \in Q, S \in \mathcal{P}(Q'), S \cap F' \neq \emptyset\} \]

**Regular languages and expressions**

Define a class of languages inductively as follows:

- $\emptyset$ is regular.
- For every $a \in \Sigma$, \{a\} is regular.
- $L_1, L_2$ regular $\implies L_1 \cup L_2$ regular.
- $L_1, L_2$ regular $\implies L_1 \cdot L_2$ regular.
- $L$ regular $\implies L^*$ regular.

Any such language described by a regular expression, such as:

\[ a(a \cup b)^* a \cup b(a \cup b)^* b \cup a \cup b \]
An application of DFAs

String Searching Problem

**Input:** string $T$ of length $n$. string $w$ of length $k$.

**Output:** Yes/No. Does $w$ occur in $T$?

Naive algorithm:
- About $nk$ steps.
- Can we do better?

Automaton solution:
- The language $\Sigma^* w \Sigma^*$ is regular.
- So there is some DFA $M_w$ that accepts it.
- Build $M_w$ and feed it $T$. Running time: $\sim n$ steps.
- Time to build $M_w$?
  - Simple alg: $\sim k^3$ steps.
  - Knuth-Morris-Pratt 1977: $\sim k$ steps to build $M_w$. 