How do you do?

Draw a DFA* that describes your typical day.

*If you know how to clone yourself OR if you’re all about the multiverse, feel free to draw an NFA that describes your typical day.
Recap: Distinguishability of two strings

Two strings \( w_1, w_2 \) are distinguishable if there is some other string \( z \) such that \( w_1 z \in L \) and \( w_2 z \notin L \).

\[
L = \{ w \mid w's \ text{ length is divisible by 3} \}
\]

1 and 11 are distinguishable:

\[
\begin{align*}
  \ w_1 &= 1 \\
  \ w_2 &= 11 \\
  \ z &= 1 \\
  \ w_1z &= 11 \notin L \\
  \ w_2z &= 111 \in L
\end{align*}
\]
Recap: Pair-wise distinguishability of a set of strings

A set $S = \{w_1, w_2, \ldots, w_n\}$ is pairwise distinguishable for a language $L$ if every pair of strings $w_i \neq w_j$ is distinguishable.

$L = \{w \mid w\text{’s length is divisible by 3}\}$

The set $\{\lambda, 1, 11\}$ is pair-wise distinguishable:

<table>
<thead>
<tr>
<th></th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>$\lambda$</td>
<td>n/a</td>
<td>11</td>
</tr>
<tr>
<td>$w_2$</td>
<td>1</td>
<td>redundant</td>
<td>n/a</td>
</tr>
<tr>
<td>$w_3$</td>
<td>11</td>
<td>redundant</td>
<td>redundant</td>
</tr>
</tbody>
</table>
Recap: Distinguishability theorem

If a set $S = \{w_1, w_2, \ldots, w_n\}$ (i.e., a set of size $n$) is pairwise distinguishable for a language $L$, then any DFA that accepts $L$ must have at least $n$ states.

$L = \{w \mid w$’s length is divisible by 3$\}$

$L$ requires at least 3 states.
Practice: Distinguishability theorem

If a set $S = \{w_1, w_2, \ldots, w_n\}$ (i.e., a set of size $n$) is pairwise distinguishable for a language $L$, then any DFA that accepts $L$ must have at least $n$ states.

$L = \{w \mid \text{the third bit in } w \text{ is a } 1\}$

$L$ requires at least 5 states.

<table>
<thead>
<tr>
<th></th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>n/a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
If a set $S = \{w_1, w_2, \ldots, w_n\}$ (i.e., a set of size $n$) is pairwise distinguishable for a language $L$, then any DFA that accepts $L$ must have at least $n$ states.

$L = \{w \mid \text{the third bit in } w \text{ is a } 1\}$

$L$ requires at least 5 states.
Are regular languages sufficient?

(1) \( L = \{ a^N b^N \mid N > 0 \} \)  
   \[ \text{equality} \]
   \[ \text{this means } N \text{ repetitions of the character `a`} \]

(2) \( L = \{ a^N b^{2N} \mid N > 0 \} \)  
   \[ \text{multiplication} \]

(3) \( L = \{ a^N b^M c^{(N+M)} \mid N,M > 0 \} \)  
   \[ \text{addition} \]

Not Regular
we cannot build a DFA that accepts \( L \)
Myhill–Nerode theorem

A language $L$ is **regular** if and only if we can define a set $S$ of pairwise distinguishable strings such that $|S|$ is **finite**.

So...how can we prove that $L$ is **not** regular?
How do we prove that languages aren’t regular?

Intuition: a language $L$ is not regular if a “DFA” for it would require an infinite number of states.

So: if we can describe an infinite set of pairwise-distinguishable strings for $L$, then $L$ is not regular.
Prove that $L = \{a^N b^N \mid N > 0\}$ is not regular.

Let $S = a a^*$

<table>
<thead>
<tr>
<th>$w_i$</th>
<th>$w_j$</th>
<th>$z$</th>
<th>Accept $w_i z$</th>
<th>Reject $w_j z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>aa</td>
<td>b</td>
<td>ab</td>
<td>aab</td>
</tr>
<tr>
<td>a</td>
<td>aaa</td>
<td>b</td>
<td>ab</td>
<td>aaab</td>
</tr>
<tr>
<td>a</td>
<td>aaaa</td>
<td>b</td>
<td>ab</td>
<td>aaaab</td>
</tr>
<tr>
<td>a</td>
<td>aaaaa</td>
<td>b</td>
<td>ab</td>
<td>aaaaab</td>
</tr>
</tbody>
</table>

...and so on for every pair of unequal strings in $S$...
When (not) to use regular languages

Regular languages are useful for
- processes that require a finite number of steps
- recognizing text that doesn’t require us to remember input

Please don’t use regular expressions to parse HTML!

Regular languages are not useful for
- modeling the full power of a computer
Where are we?

We wanted to come up with a precise definition for the word *computer*.

We’ve got a candidate definition: *automata that recognize regular languages (DFAs/NFAs)*.

We want to know how good this definition is.

It’s not good enough.
Deterministic Finite Automaton

Formal definition

A machine $M$ that consists of:

- an alphabet $\Sigma$
- a finite set of states, including:
  - initial state
  - accepting state(s)
- transitions between states
  for every state, every letter in $\Sigma$ labels one and only one transition

Given a string $w$, $M$ accepts $w$ if consuming $w$ causes $M$ to terminate in an accepting state.

what’s missing?
Turing Machines

Formal definition

A machine $M$ that consists of:

- an alphabet $\Sigma$
- a finite set of states, including:
  - initial state
  - accepting state(s)
- transitions between states
- an infinitely large tape, which can be read or written
  the tape is akin to memory
- a current location on the tape
called the “read/write head”

Given a string $w$, $M$ accepts $w$ if consuming $w$ causes $M$ to terminate in an accepting state.
Turing Machines

- Tape
- R/W head
- Finite-state "controller"
- Can move left or right

- Blank space, spelled ☐

Diagram:

- CS 60

Note: The text and diagram illustrate the components of a Turing machine, including the tape, R/W head, finite-state controller, and capability to move left or right.
can move left or right

https://youtu.be/E3keLeMwfHY
A Turing machine

- Tape
- R/W head
- Blank space, spelled □

One transition

"If I see a C...

"...write a C...

"...and move right...

C ; C, R

A controller

Rewrites "CS 60" to "CS 42"
Let’s practice!

What does this machine do for the input 11?
What does this machine do in general?
$\Sigma = \{a, b\}$

$L = \{a^N b^N\}$

**Bonus problem!**
\[ \Sigma = \{a, b\} \quad L = \{a^N b^N\} \]
Turing Machines FTW!

(1) \( L = \{a^N b^N \mid N > 0\} \)  // equality

(2) \( L = \{a^N b^{2N} \mid N > 0\} \)  // multiplication

(3) \( L = \{a^N b^M c^{(N+M)} \mid N, M > 0\} \)  // addition

So far, all known computational devices are equivalent to Turing Machines...

Quantum computers
Molecular computers
Parallel computers
Integrated circuits
Water-based computation
...
Turing Machines FTW?

Here’s a strategy for doing every HW assignment in every class:

(1) Spend a week writing a program that takes a description of any assignment and computes the solution.

(2) There is no step two.
We must know.
We will know.
We must know.
We will know.

No.
Kurt Gödel
This sentence is false.
Given a computational problem
Is there a solution?
What is it?
How good is it?

Not always! Some problems are undecidable.
The Halting Problem is undecidable

$L = \{ \text{All programs that halt and give an answer} \}$
The Halting Problem is undecidable

$L = \{\text{All programs that halt and give an answer}\}$

Proof sketch (proof by contradiction):

1. Assume that Halting Problem is decidable.
2. Show that this assumption leads to a contradiction.
3. Therefore the assumption (that the HP is decidable) is false.
The Halting Problem is undecidable

L = \{All programs that halt and give an answer\}

1. Assume that Halting Problem is decidable.

\[ M_{HP} \]
The Halting Problem is undecidable

\[ L = \{ \text{All programs that halt and give an answer} \} \]

1. Assume that Halting Problem is decidable.
2. Show that this assumption leads to a contradiction.
The Halting Problem is undecidable

$L = \{\text{All programs that halt and give an answer}\}$

1. Assume that Halting Problem is decidable.
2. Show that this assumption leads to a contradiction.
The Halting Problem is undecidable

\[ L = \{ \text{All programs that halt and give an answer} \} \]

1. Assume that Halting Problem is decidable.
2. Show that this assumption leads to a contradiction.
The Halting Problem is undecidable

$L = \{\text{All programs that halt and give an answer}\}$

1. Assume that Halting Problem is decidable.
2. Show that this assumption leads to a contradiction.
The Halting Problem is undecidable

\[ L = \{ \text{All programs that halt and give an answer} \} \]

1. Assume that Halting Problem is decidable.
2. Show that this assumption leads to a contradiction.
Programs $\equiv$ Data
What good is Computer Science?