The Big Picture
Announcements

- **Final exam** this Friday from 12:15PM-3:15PM
  - Please let us know *immediately after lecture* if you want to take the final at an alternate time and haven't yet told us.

- Problem Set 9 due right now. Congrats on finishing the last problem set of the quarter!

- Problem Set 7 graded and available for pickup.
The Big Picture
The Big Picture
Imagine what it must have been like to discover all of the results in this class.
Cantor's Theorem: $|S| < |\mathcal{P}(S)|$

Corollary: There exist unsolvable problems.
What problems are unsolvable?
First, we need to learn how to prove things.

Otherwise, how do we know for sure that we're right?
Now, we need to learn how to prove things about processes that proceed step-by-step.

So let's learn induction.
We also should be sure we have some rules about how reasoning works.

Let's add some logic into the mix.
Okay! So now we're ready to go.

What problems are unsolvable?
Well... first we need a definition of a computer!
Cool! Now we have a model of a computer!
We're not quite sure what we can solve at this point, but that's okay for now.

Let's call the languages we can capture this way the **regular languages**.
I wonder what other machines we can make?
Wow! Those new machines are way cooler than our old ones!
I wonder if they're more powerful?
Wow! I guess not. That's surprising!

So now we have a new way of modeling computers with finite memory!
I wonder how I can combine these machines together?
Cool! Since we can glue machines together, I guess we can glue languages together as well!
How are we going to do that?
a^+(.a^+)^*@a^+(.a^+)^+
Cool! We've got a new way of describing languages.
So what sorts of languages can we describe?
Awesome! We got back the exact same class of languages.
It seems like all our models give us the same power! Did we get every language?
Wow! I guess not.
But we did learn something cool:

We have just explored what problems can be solved with finite computers.
So what else is out there?
Well... what if we add memory to our machines?
$$\text{int}, \varepsilon \rightarrow \varepsilon$$

$$\text{+, } \varepsilon \rightarrow \varepsilon$$

$$\text{*}, \varepsilon \rightarrow \varepsilon$$

$$\text{, } \varepsilon \rightarrow ($$

$$\text{), } (\rightarrow \varepsilon$$

$$\varepsilon, Z_0 \rightarrow \varepsilon$$
These machines can do more than our old machines!
Can we describe these languages another way?
S → 1S1
S → 1S
S → ≥

$\varepsilon, S \rightarrow 1S$
$\varepsilon, S \rightarrow 1S1$
$\varepsilon, S \rightarrow ≥$
$\Sigma, \Sigma \rightarrow \varepsilon$

start

$\varepsilon, \varepsilon \rightarrow S$

$\varepsilon, Z_0 \rightarrow Z_0$
Awesome!
So did we get every language yet?
$uv^2xy^2z \in L$
Hmmm... guess not.
So what if we make our memory a little better?
B → B, R
0 → 0, R
1 → 1, R

0 → B, L
1 → 0, L
1 → 1, L

B → B, R
0 → 0, R
1 → 1, R

B → B, L
0 → 0, R
1 → 1, R

q_0
start
q_{acc}

q_1
q_2
q_3
q_4
q_5

q_{rej}
Wow, these are hard to design.

Is there an easier way?
// Start
0: If reading 0, go to M0.
1: If reading 1, go to M1.
2: Accept

// M0
3: Write B.
4: Move right.
5: If reading 0, go to 4.
6: If reading 1, go to 4.
7: Move left.
8: If reading 0, go to Next.
9: Reject.

// M1
10: Write B.
11: Move right.
12: If reading 0, go to 11.
13: If reading 1, go to 11.
14: Move left.
15: If reading 1, go to Next.
16: Reject.

// Next
17: Write B.
18: Move left.
19: If reading 0, go to 18
20: If reading 1, go to 18
21: Move right
22: Go to Start.
0: If reading B, go to 4.
1: If reading 1, go to 5.
2: Move right.
3: Go to 0.
4: Accept.
5: Reject.
Much better! So let's add some new features.
// Match

7: Move tape 2 left until {B}
8: Move tape 2 right.
9: Move tape 1 right.
10: Write $ to tape 1, track 2.
11: If B on tape 2, go to Acc.
12: If B on tape 1, go to Rej.
13: Load tape 1, track 1 into X.
14: Load tape 2 into Y.
15: If X = Y, go to 17.
16: Go to Mismatch.
17: Move tape 1 right.
18: Move tape 2 right.
19: Go to 11.

// Mismatch

20: Move tape 1.2 left until {\$}
21: Go to Match.

// Acc

22: Accept.

// Rej

23: Reject.
Wow! Looks like we can't make this any more powerful.

(The Church-Turing thesis says that this isn't a coincidence!)
So why is that?
Simulated tape of the program being executed.

> 0 1 x 0 0 A 0 < ...

Program tape holding the program being executed.

> 0 : Move left . 1 : Go ... 

Scratch tape for intermediate computation.

Variables for intermediate storage.

Instr    Letter
Wow! Our machines can simulate one another!

This is a theoretical justification for why all these models are equivalent to one another.
So... can we solve everything yet?
<table>
<thead>
<tr>
<th>(\langle M_0 \rangle)</th>
<th>(\langle M_1 \rangle)</th>
<th>(\langle M_2 \rangle)</th>
<th>(\langle M_3 \rangle)</th>
<th>(\langle M_4 \rangle)</th>
<th>(\langle M_5 \rangle)</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M_0)</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>(M_1)</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>(M_2)</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>(M_3)</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>(M_4)</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>(M_5)</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

No | No | No | Acc | No | Acc | ...
Oh great. Some things are impossible.
So is there just one problem we can't solve?
\[ L_D \leq M \bar{A}_{\text{TM}} \]

\[ A_{\text{TM}} \in \text{RE} \]

\[ A_{\text{TM}} \notin R \]
Okay... maybe we can't decide or recognize everything.

Can we at least verify or refute everything?
\[ L_D \leq_{\text{M}} \text{REGULAR}_{\text{TM}} \]

\[ \overline{L}_D \leq_{\text{M}} \text{REGULAR}_{\text{TM}} \]
Wow. That's pretty deep.
So what can we do efficiently?
So... how are you two related again?
No clue.
But we do we know about them?
Theory is all about exploring and experimenting.

We've barely scratched the surface of what we can do with computers.
Theory is all about exploring and experimenting.

We've barely scratched the surface of what we can do with computers.
Where to go from here?
CS154

- Intro to Automata and Complexity Theory
- An in-depth treatment of automata, computability, and complexity.
- Emphasis on theoretical results in automata theory and complexity.
- Launching point for more advanced courses (CS254, CS354)
CS258

- Intro to Programming Language Theory
- Explore questions of computability in terms of recursion and recursive functions.
- Excellent complement to the material in this course; highly recommended.
- Offered every other year; consider checking it out when it comes around next time!
CS109

- Intro to Probability for Computer Scientists
- Learn to embrace randomness.
- Use your newly acquired proof skills in an entirely different domain.
- Learn why Mehran Sahami is so awesome.
CS255

- Intro to Cryptography
- How can you use hard problems to your advantage?
- How can $\textbf{NP}$-completeness help you keep secrets?
CS161

- Design and Analysis of Algorithms
- Learn how to approach new problems and solve them efficiently.
- Learn how to deal with $\text{NP}$-completeness.
- Learn how to ace job interviews
CS143

- **Compilers**
- Watch automata, grammars, undecidability, and \textbf{NP}-completeness come to life by building a complete working compiler from scratch.
- See just how much firepower you can get from all this material.
CS107

- Computer Organization and Systems
- You don't need to be a theoretician to love computer science!
- If you want to learn how the machine works under the hood, look no further.
There are more problems to solve than there are programs to solve them.
Where We've Been

- Given this hard theoretical limit, what can we compute?
- How powerful of a computer do we need to reach this limit?
- Of what we can compute, what can we compute efficiently?
- What techniques from mathematics can we use to reason about this?
What We've Covered

- Sets
- Graphs
- Proof Techniques
- Relations
- Induction
- Logic
- Pigeonhole Principle
- SAT Solvers
- DFAs
- NFAs
- Regular Expressions

- CFGs
- PDAs
- Pumping Lemmas
- Turing Machines
- Undecidability
- Unrecognizability
- Reductions
- Time Complexity
- \( P \)
- \( NP \)
- \( NP \)-Completeness
Final Thoughts