Hello Condensed Slide Readers!

Today’s lecture consists almost exclusively of animations of Turing machines and TM constructions. We’ve presented a condensed version here, but we strongly recommend reading the full version of the slides today.

Hope this helps!

—Keith
What problems can we solve with a computer?
Languages recognizable by any feasible computing machine
That same drawing, to scale.
The Problem

- Finite automata accept precisely the regular languages.
- We may need unbounded memory to recognize context-free languages.
  - e.g. \( \{ a^n b^n \mid n \in \mathbb{N} \} \) requires unbounded counting.
- How do we build an automaton with finitely many states but unbounded memory?
A Brief History Lesson
A Simple Turing Machine

This is the Turing machine’s **finite state control**. It issues commands that drive the operation of the machine.
A Simple Turing Machine

This is the TM’s *infinite tape*. Each tape cell holds a *tape symbol*. Initially, all (infinitely many) tape symbols are blank.

...
The machine is started with the **input string** written somewhere on the tape. The **tape head** initially points to the first symbol of the input string.

```
... a a a a a ...
```
A Simple Turing Machine

Like DFAs and NFAs, TMs begin execution in their start state.
A Simple Turing Machine

At each step, the TM only looks at the symbol immediately under the tape head.
A Simple Turing Machine

These two transitions originate at the current state. We’re going to choose one of them to follow.
A Simple Turing Machine

Each transition has the form

\[ \text{read} \rightarrow \text{write}, \text{dir} \]

and means “if symbol \text{read} is under the tape head, replace it with \text{write} and move the tape head in direction \text{dir} (L or R). The \(\square\) symbol denotes a blank cell.
A Simple Turing Machine

Each transition has the form

\[ \text{read } \rightarrow \text{ write, dir} \]

and means “if symbol \textbf{read} is under the tape head, replace it with \textbf{write} and move the tape head in direction \textbf{dir} (L or R). The \(\square\) symbol denotes a blank cell.
A Simple Turing Machine

1. Start in state $q_0$.
2. If the symbol is $\square$, move right, stay in state $q_0$.
3. If the symbol is $a$, move right, go to state $q_1$.
4. If the symbol is $\square$, move right, go to state $q_{\text{rej}}$.
5. If the symbol is $a$, move right, go to state $q_{\text{acc}}$.

 transitions:
- $\square \rightarrow \square, R$
- $a \rightarrow \square, R$
- $a \rightarrow a, R$
- $\square \rightarrow \square, R$

States:
- $q_0$
- $q_1$
- $q_{\text{rej}}$
- $q_{\text{acc}}$

Input:
- ... a a a a ...

Output:
A Simple Turing Machine

Unlike a DFA or NFA, a TM doesn’t stop after reading all the input characters. We keep running until the machine explicitly says to stop.
A Simple Turing Machine

This special state is an accepting state. When a TM enters an accepting state, it immediately stops running and accepts whatever the original input string was (in this case, aaaa).
A Simple Turing Machine

This special state is a **rejecting state**. When a TM enters a rejecting state, it **immediately** stops running and rejects whatever the original input string was (in this case, `aaaaa`).
A Simple Turing Machine

If the TM is started on the empty string \( \varepsilon \), the entire tape is blank and the tape head is positioned at some arbitrary location on the tape.
The Turing Machine

• A Turing machine consists of three parts:
  • A *finite-state control* that issues commands,
  • an *infinite tape* for input and scratch space, and
  • a *tape head* that can read and write a single tape cell.

• At each step, the Turing machine
  • writes a symbol to the tape cell under the tape head,
  • changes state, and
  • moves the tape head to the left or to the right.
Input and Tape Alphabets

- A Turing machine has two alphabets:
  - An **input alphabet** Σ. All input strings are written in the input alphabet.
  - A **tape alphabet** Γ, where Σ ⊆ Γ. The tape alphabet contains all symbols that can be written onto the tape.

- The tape alphabet Γ can contain any number of symbols, but always contains at least one **blank symbol**, denoted □. You are guaranteed □ ∉ Σ.

- At startup, the Turing machine begins with an infinite tape of □ symbols with the input written at some location. The tape head is positioned at the start of the input.
Accepting and Rejecting States

- Unlike DFAs, Turing machines do not stop processing the input when they finish reading it.
- Turing machines decide when (and if!) they will accept or reject their input.
- Turing machines can enter infinite loops and never accept or reject; more on that later...
Determinism

- Turing machines are **deterministic**: for every combination of a (non-accepting, non-rejecting) state $q$ and a tape symbol $a \in \Gamma$, there must be exactly one transition defined for that combination of $q$ and $a$.

- Any transitions that are missing implicitly go straight to a rejecting state. We’ll use this later to simplify our designs.
Determinism

- Turing machines are **deterministic**: for every combination of a (non-accepting, non-rejecting) state $q$ and a tape symbol $a \in \Gamma$, there must be exactly one transition defined for that combination of $q$ and $a$.

- Any transitions that are missing implicitly go straight to a rejecting state. We’ll use this later to simplify our designs.

![Diagram of Turing machine]

This machine is exactly the same as the previous one.
Run the TM shown above on the input string $bba$. What will the tape look like when the TM finishes running?
The given diagram represents a finite automaton with the following transitions:

- From state $q_0$, on input $a$, go to state $q_1$ and move right ($R$).
- From state $q_0$, on input $b$, go to state $q_1$ and move right ($R$).
- From state $q_1$, on input $\square$, go to state $q_1$ and move left ($L$).
- From state $q_1$, on input $a$, go to state $q_{acc}$ and move left ($L$).
- From state $q_1$, on input $b$, go to state $q_{rej}$ and move left ($L$).

The automaton starts at state $q_0$ and accepts strings containing exactly two 'a's, one 'b', and any number of 'a's on the left. It rejects any other input.
If \( M \) is a Turing machine with input alphabet \( \Sigma \), then the *language of \( M \)*, denoted \( \mathcal{L}(M) \), is the set

\[
\mathcal{L}(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}
\]

What is \( \mathcal{L}(M) \), where \( M \) is the above TM?
Although the tape ends with $bba$ written on it, the original input string was $aab$. This shows that the TM accepts $aab$, not $bba$.

So $\mathcal{L}(M) = \{ w \in \{a, b\}^* \mid w \text{ ends in } b \}$
Designing Turing Machines

- Despite their simplicity, Turing machines are very powerful computing devices.
- Today's lecture explores how to design Turing machines for various languages.
Designing Turing Machines

- Let $\Sigma = \{0, 1\}$ and consider the language $L = \{0^n1^n | n \in \mathbb{N}\}$.
- We know that $L$ is context-free.
- How might we build a Turing machine for it?
\[ L = \{ 0^n1^n \mid n \in \mathbb{N} \} \]
A Recursive Approach

- The string $\varepsilon$ is in $L$.
- The string $0w1$ is in $L$ iff $w$ is in $L$.
- Any string starting with $1$ is not in $L$.
- Any string ending with $0$ is not in $L$. 
\[ q_{\text{acc}} \rightarrow \square, R \]
\[ q_{\text{rej}} \rightarrow \square, R \]

### States:
- **Start**: 0 → 0, L
- **Check for 0**: 1 → 0, R
- **Clear a 1**: 1 → 1, L
- **Go to Start**: 0 → 0, R
- **Go to End**: 0 → 0, L
- **Go to End**: 1 → 1, R

### Transitions:
- **Start**:
  - 0 → 0, L
  - 1 → 1, L

- **Check for 0**:
  - 1 → 0, R

- **Clear a 1**:
  - 1 → 1, L

- **Go to End**:
  - 0 → 0, R
  - 1 → 1, R
Another TM Design

• We've designed a TM for \( \{0^n1^n \mid n \in \mathbb{N}\} \).

• Consider this language over \( \Sigma = \{0, 1\} \):

\[
L = \{ w \in \Sigma^* \mid w \text{ has the same number of } 0\text{s and } 1\text{s} \}
\]

• This language is also not regular, but it is context-free.

• How might we design a TM for it?
A Caveat

How do we know that this blank isn't one of the infinitely many blanks after our input string?
One Solution

... × 0 0 × 1 1 1 0 ...

...
Remember that all missing transitions implicitly reject.
Constant Storage

- Sometimes, a TM needs to remember some additional information that can't be put on the tape.
- In this case, you can use similar techniques from DFAs and introduce extra states into the TM's finite-state control.
- The finite-state control can only remember one of finitely many things, but that might be all that you need!
Time-Out for Announcements!
Problem Set Seven

- Problem Set Seven is due this Friday at 2:30PM.
  - As always, if you have questions, feel free to stop by office hours or ask on Piazza!

- As a heads-up, Problem Set Eight will go out on Friday. It’ll be due the Friday after break, so you’ll have some time to decompress. 😊
Your Questions
“Can you guys upvote this if you legit think you failed the midterm yesterday cause I’m freaking out”

“same here, also nervous of failing the class, welp!”

“Can you guys upvote this if you legit think you failed the midterm yesterday cause I’m freaking out”

“same here, also nervous of failing the class, welp!”

I’m scared of failing this class, but I’d hate to withdraw after coming all this way. Where’s the line between saving grades and not giving up before the end?”

“same here, also nervous of failing the class, welp!”

What was the median for PSET 4, 5 and 6? Please :) I want to know that I'm not falling behind. The induction PSET was SO HARD.

How do i know if I should take this class credit/no credit or for a grade? How kind is the curve?

Is there a way we could see the overall median at this point so we can decide whether to withdraw/switch to credit no credit?

How do i know if I should take this class credit/no credit or for a grade? How kind is the curve?

What was the median for PSET 4, 5 and 6? Please :) I want to know that I'm not falling behind. The induction PSET was SO HARD.
Don’t panic.
Use math.
Preliminary Exam Solutions

• We’ve posted a set of preliminary exam solutions to the course website.

• It contains
  – solutions to all the exam questions, or at least, one set of solutions;
  – a recap about how to compute your raw score so far and how to extrapolate;
  – grade cutoffs from past quarters; and
  – statistics on all the problem sets.

• Remember: 44% of your grade is still completely under your control at this point. Think about your ability to grow, not where you currently stand.
Back to CS103!
Another TM Design

• We just designed a TM for this language over $\Sigma = \{0, 1\}$:

$$L = \{ w \in \Sigma^* | \text{w has the same number of 0s and 1s}\}$$

• Let's do a quick review of how it worked.
The Solution

... × 0 0 × 1 1 1 1 0 ...

...
A Different Idea
A Different Strategy

Could we sort the characters of this string?
A Different Strategy

Observation 1: A string of 0s and 1s is sorted if it matches the regex $0^*1^*$. 
A Different Strategy

Observation 2: A string of 0s and 1s is not sorted if it contains 10 as a substring.
A Different Strategy

Idea: Repeatedly find a copy of 10 and replace it with 01.
Let's Build It!
Our ultimate goal here was to sort everything so we could hand it off to the machine to check for $0^n 1^n$. Let’s rewind the tape head back to the start.
This is just a placeholder. Imagine snapping in the entire TM for $0^n1^n$ into this diagram, putting the start state in the dashed area.
This TM will sort any sequence of 0s and 1s, but it might take a while.

**Fun problem:** design a TM that sorts a string of 0s and 1s, but does so while taking way fewer steps than this machine.
TM Subroutines

- A **TM subroutine** is a Turing machine that, instead of accepting or rejecting an input, does some sort of processing job.
- TM subroutines let us compose larger TMs out of smaller TMs, just as you'd write a larger program using lots of smaller helper functions.
- Here, we saw a TM subroutine that sorts a sequence of 0s and 1s into ascending order.
TM Subroutines

• Typically, when a subroutine is done running, you have it enter a state marked “done” with a dashed line around it.

• When we're composing multiple subroutines together – which we'll do in a bit – the idea is that we'll snap in some real state for the “done” state.
What other subroutines can we make?