Turing Machines
Part One
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?
Regular Languages

CFLs

Languages recognizable by any feasible computing machine

All Languages
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 | n \in \mathbb{N} \} requires unbounded counting.
- How do we build an automaton with finitely many states but unbounded memory?
A Brief History Lesson
\[
\begin{array}{cccccccccccc}
2 & 7 & 1 & 8 & 2 & 8 & 1 & 8 & 2 & 8 & 4 & 5 & 9 & 0 \\
+ & 3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 & 5 & 3 & 5 & 8 & 9 & 7 \\
\hline
5 & 8 & 5 & 9 & 8 & 7 & 4 & 4 & 8 & 2 & 0 & 4 & 8 & 7 \\
\end{array}
\]
Modeling This Idea: *Turing Machines*
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.
A Simple Turing Machine

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.

The tape initially contains the string **"… a a a a a a …"**.
Like DFAs and NFAs, TMs begin execution in their start state.
At each step, the TM only looks at the symbol immediately under the tape head.
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 □ symbol denotes a blank cell.
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

\[ q_0 \rightarrow a, R \]

\[ a \rightarrow \square, R \]

\[ \square \rightarrow \square, R \]

\[ q_0 \rightarrow q_1 \]

\[ q_1 \rightarrow q_{\text{rej}} \]

\[ q_{\text{rej}} \rightarrow q_{\text{acc}} \]

\[ \text{start} \]

... a a a a ...

...
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 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

$q_{\text{acc}}$

$\square \rightarrow \square, R$

$a \rightarrow \square, R$

$q_0$

$\begin{array}{c}
\text{start} \\
\rightarrow \end{array}$

$q_1$

$a \rightarrow \square, R$

$q_{\text{rej}}$

$\square \rightarrow \square, R$

$a \rightarrow \square, R$

$q_{\text{rej}}$

... a a a a a a a ...
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 \textit{finite-state control} that issues commands,
  • an \textit{infinite tape} for input and scratch space, and
  • a \textit{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* \( \Sigma \). All input strings are written in the input alphabet.
  - A *tape alphabet* \( \Gamma \), where \( \Sigma \subseteq \Gamma \). The tape alphabet contains all symbols that can be written onto the tape.
  - The tape alphabet \( \Gamma \) can contain any number of symbols, but always contains at least one *blank symbol*, denoted \( \square \). You are guaranteed \( \square \notin \Sigma \).
  - At startup, the Turing machine begins with an infinite tape of \( \square \) 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](image-url)
Run the TM shown above on the input string bba. What will the tape look like when the TM finishes running?
\[ a \rightarrow b, \text{R} \]
\[ b \rightarrow a, \text{R} \]
\[ a \rightarrow a, \text{L} \]
\[ b \rightarrow b, \text{L} \]

Start state: \( q_0 \)

Transitions:
- \( a \rightarrow b, \text{R} \) from \( q_0 \) to \( q_1 \)
- \( b \rightarrow a, \text{R} \) from \( q_0 \) to \( q_1 \)
- \( a \rightarrow a, \text{L} \) from \( q_1 \) to \( q_1 \)
- \( b \rightarrow b, \text{L} \) from \( q_1 \) to \( q_1 \)
- \( \square \rightarrow \square, \text{L} \) from \( q_1 \) to \( q_1 \)

Accepting state: \( q_{\text{acc}} \)

Rejecting state: \( q_{\text{rej}} \)

Input string: \( \ldots a a b \ldots \)
If $M$ is a Turing machine with input alphabet $\Sigma$, then the \textit{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 \( \mathcal{L}(M) = \{ w \in \{a, b\}^* | w \text{ ends in } b \} \)
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 $L = \{0^n1^n \mid n \in \mathbb{N}\}$.

• 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 $L$. 
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?
Remember that all missing transitions implicitly reject.
Time-Out for Announcements!
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.
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!

• We’re working to get Problem Set Six returned ASAP; stay tuned!
Your Questions
“Hey Keith, do you ever think, "You know what? These guys took a midterm this week and did great. How about no homework!" Wouldn't that be nice."

I’m aware that there’s a lot going on in the quarter right now, and I wanted to thank all of you for putting in the effort and rising to the occasion. Hang in there - we’re really impressed by how far you’ve all come!
Back to CS103!
Another TM Design

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

\[
L = \{0^n1^m \mid n, m \in \mathbb{N} \text{ and } m \text{ is a multiple of } n \}
\]

• Is this language regular?

• How might we design a TM for this language?
An Observation

• We can recursively describe when one number $m$ is a multiple of $n$:
  - If $m = 0$, then $m$ is a multiple of $n$.
  - Otherwise, if $n = 0$, then $m$ is not a multiple of $n$.
  - Otherwise, $m$ is a multiple of $n$ iff $m \geq n$ and $m - n$ is a multiple of $n$.

• Idea: Repeatedly subtract $n$ from $m$ until $m$ becomes zero (good!) or drops below zero (bad!)
Concepts from Today

• Turing machines are a generalization of finite automata equipped with an infinite tape.
• It's often helpful to think recursively when designing Turing machines.
• It's often helpful to introduce new symbols into the tape alphabet.
• Watch for edge cases that might lead to infinite loops – though we'll say more about that later on.
Next Time

- **TM Subroutines**
  - Combining multiple TMs together!
- **The Church-Turing Thesis**
  - Just how powerful are Turing machines?
- **R and RE Languages**
  - What does it mean to solve a problem?