Turing Machines
Part Three
What problems can we solve with a computer?
Very Important Terminology

- Let $M$ be a Turing machine.
- $M$ accepts a string $w$ if it enters an accept state when run on $w$.
- $M$ rejects a string $w$ if it enters a reject state when run on $w$.
- $M$ loops infinitely (or just loops) on a string $w$ if when run on $w$ it enters neither an accept nor a reject state.
- $M$ does not accept $w$ if it either rejects $w$ or loops infinitely on $w$.
- $M$ does not reject $w$ if it either accepts $w$ or loops on $w$.
- $M$ halts on $w$ if it accepts $w$ or rejects $w$. 

\[ \begin{align*}
\text{Accept} & \quad \text{does not reject} \\
\text{Loop} & \quad \text{does not accept} \\
\text{Reject} & \quad \text{halts}
\end{align*} \]
**Recognizable Languages (RE)**

- The language of a Turing machine $M$, denoted $\mathcal{L}(M)$, is the set of all strings that $M$ accepts:
  \[ \mathcal{L}(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \} \]

- For any $w \in \mathcal{L}(M)$, $M$ accepts $w$.
- For any $w \notin \mathcal{L}(M)$, $M$ does not accept $w$.
  - $M$ might reject, or it might loop forever.
- A language is called **recognizable** if it is the language of some TM.
- A TM $M$ where $\mathcal{L}(M) = L$ is called a **recognizer** for $L$.
- Notation: the class $\textbf{RE}$ is the set of all recognizable languages.

\[ L \in \textbf{RE} \iff L \text{ is recognizable} \]
Decidable Languages (R)

- Is it possible to write a TM that is guaranteed never to loop regardless of input? Yes, such TMs do exist. They’re called **deciders**.

- A language $L$ is called **decidable** if there exists a decider $M$ such that $\mathcal{L}(M) = L$.

- Equivalently, a language $L$ is decidable if there is a TM $M$ such that
  - If $w \in L$, then $M$ accepts $w$.
  - If $w \notin L$, then $M$ rejects $w$.

- The class $\mathbf{R}$ is the set of all decidable languages.

$$L \in \mathbf{R} \iff L \text{ is decidable}$$
R and RE Languages

• Every decider is a Turing machine, but not every Turing machine is a decider.

• This means that $\mathbb{R} \subseteq \mathbb{RE}$.

• But is it a strict subset?

• That is, if you can just confirm “yes” answers to a problem, can you necessarily solve that problem?
Which Picture is Correct?

- Regular Languages
- CFLs
- RE

All Languages
Which Picture is Correct?
To answer the question is $R$ a *strict* subset of $RE$, I need to quickly convince you of three little things.
1. We can give a machine all kinds of inputs as a string, including groups of inputs and machines themselves!
1. We can give a machine all kinds of inputs as a string, including groups of inputs and machines themselves!

2. There is a TM that can take TMs as input and run them to see what they do. ("Universality" property)

3. That TM, or any TM that takes other TMs as input, could take itself as input. ("Self-Reference" property)
A Model for Solving Problems

Turing Machine

How do we represent our inputs?

input

Yes
(accept)

No
(reject)
On your computer, *everything* is numbers!

- Images (gif, jpg, png):
- Integers (int):
- Non-integer real numbers (double):
- Letters and words (ASCII, Unicode):
- Music (mp3):
- Movies (streaming):
- Doge pictures:
- Email messages:
Object Encodings

- If \( \text{Obj} \) is some mathematical object that is \textit{discrete} and \textit{finite}, then we’ll use the notation \langle \text{Obj} \rangle to refer to some way of encoding that object as a string.

- Think of \langle \text{Obj} \rangle like a file on disk – it encodes some high-level object as a series of characters.

\[ \langle \text{Obj} \rangle = 11011100101110111100010011\ldots110 \]
Object Encodings

• For the purposes of what we’re going to be doing, we aren’t going to worry about exactly how objects are encoded.

• For example, we could say \(G_{0n1n}\) to mean “some encoding of a Context-Free Grammar for the language \(0^n1^n\)” without worrying about exactly how a grammar is encoded.
  
  • Intuition check: could I type up any grammar and save it as a file on my computer? Yes? Ok then, we know we can put a grammar into a string.

• As long as we’re convinced a thing could be saved in a file, we don’t spend time specifying the exact file format in our proof (some proofs do in certain circumstances where it might be questioned where it’s possible, but in this class we won’t).
Encoding Groups of Objects

- Given a group of objects $Obj_1, Obj_2, \ldots, Obj_n$, we can create a single string encoding all these objects.
  - Think of it like a .zip file, but without the compression.
- We'll denote the encoding of all of these objects as a single string by $\langle Obj_1, \ldots, Obj_n \rangle$.
- This lets us feed a group of inputs into our computational device as a single input.
2. There is a TM that can take TMs as input and run them to see what they do.
An Observation

- When we've been discussing Turing machines, we've talked about designing specific TMs to solve specific problems.
- Does this match your real-world experiences? Do you have one computing device for each task you need to perform?
Computers and Programs

- Most real computers solve a variety of problems.
- To solve new problems, we get new apps, not a new computer.
- **Question**: Can we do something like this for Turing machines?
Can there be a program that simulates what another program does?

• Sure.

• These programs go by many names:
  - An **interpreter**, like the Java Virtual Machine or most implementations of Python.
  - A **virtual machine**, like VMWare or VirtualBox, that simulates an entire computer.
Can there be a TM that simulates what another TM does?

- Sure.
- You could imagine a TM taking a TM table like this as input, and then seeing how it would perform on various sample inputs by simulating its behavior by referring to the table at each step.

<table>
<thead>
<tr>
<th>State (q)</th>
<th>0</th>
<th>1</th>
<th>□</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_1$</td>
<td>□</td>
<td>R</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_1$</td>
<td>0</td>
<td>R</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_r$</td>
<td>0</td>
<td>R</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_3$</td>
<td>0</td>
<td>L</td>
</tr>
</tbody>
</table>
A TM Simulator

- It is possible to program a TM simulator on an unbounded-memory computer.
  - In fact, we did this! It's on the CS103 website.
- We could imagine it as a method

  ```java
  boolean simulateTM(TM M, string w)
  ```

  with the following behavior:
  - If $M$ accepts $w$, then `simulateTM(M, w)` returns `true`.
  - If $M$ rejects $w$, then `simulateTM(M, w)` returns `false`.
  - If $M$ loops on $w$, then `simulateTM(M, w)` loops infinitely.
The Universal Turing Machine

- **Theorem (Turing, 1936):** There is a Turing machine $U_{\text{TM}}$ called the **universal Turing machine** that, when run on an input of the form $\langle M, w \rangle$, where $M$ is a Turing machine and $w$ is a string, simulates $M$ running on $w$ and does whatever $M$ does on $w$ (accepts, rejects, or loops).

- The observable behavior of $U_{\text{TM}}$ is the following:
  - If $M$ accepts $w$, then $U_{\text{TM}}$ accepts $\langle M, w \rangle$.
  - If $M$ rejects $w$, then $U_{\text{TM}}$ rejects $\langle M, w \rangle$.
  - If $M$ loops on $w$, then $U_{\text{TM}}$ loops on $\langle M, w \rangle$.

- $U_{\text{TM}}$ accepts $\langle M, w \rangle$ if and only if $M$ accepts $w$. 
A Universal Machine

$U_{\text{TM}}$
A Universal Machine

$U_{TM}$

... program ...
A Universal Machine

$U_{TM}$

... program input ...

...
A Universal Machine

The "program" is an encoding of some Turing machine $M$ that we want to run.
A Universal Machine

$U_{TM}$

The input to that program is some string
A Universal Machine

The input has the form \( \langle M, w \rangle \), where \( M \) is some TM and \( w \) is some string.
A Universal Machine
Let $M$ be the TM shown here. How many of the following statements are true?

$M$ accepts $aa$
$U_{TM}$ accepts $\langle M, aa \rangle$
$U_{TM}$ accepts $\langle U_{TM}, \langle M, aa \rangle \rangle$
$U_{TM}$ accepts $\langle U_{TM}, \langle U_{TM}, \langle M, aa \rangle \rangle \rangle$

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then a number.
An Intuition for $U^\text{TM}$

- You can think of $U^\text{TM}$ as a general-purpose, programmable computer.

- Rather than purchasing one TM for each language, just purchase $U^\text{TM}$ and program in the “software” corresponding to the TM you actually want.

- $U^\text{TM}$ is a powerful machine: *it can perform any computation that could be performed by any feasible computing device!*
Since $U_{\text{TM}}$ is a TM, it has a language.

What is the language of the universal Turing machine?
The Language of $U_{TM}$

- Recall: For any TM $M$, the language of $M$, denoted $\mathcal{L}(M)$, is the set
  \[ \mathcal{L}(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \} \]

- What is the language of $U_{TM}$?

- $U_{TM}$ accepts $\langle M, w \rangle$ iff $M$ is a TM that accepts $w$.

- Therefore:
  \[ \mathcal{L}(U_{TM}) = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \} \]

- For simplicity, define $A_{TM} = \mathcal{L}(U_{TM})$. This is an important language and we'll see it many times.
Let $M$ be a TM where $\mathcal{L}(M) = \{ a^n b^n \mid n \in \mathbb{N} \}$

How many of the following statements are true?

$\langle M, \varepsilon \rangle \in A_{TM}$
$\langle M, a \rangle \in A_{TM}$
$\langle M, b \rangle \in A_{TM}$
$\langle M, ab \rangle \in A_{TM}$

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then a number.
3. That TM, or any TM that takes other TMs as input, could take itself as input. ("Self-Reference" property)
Getting more specific about equivalence of TMs and programs

- Every TM
  - receives some input,
  - does some work, then
  - (optionally) accepts or rejects.
- We can model a TM as a computer program where
  - the input is provided by a special method `getInput()` that returns the input to the program,
  - the program's logic is written in a normal programming language, and
  - the program (optionally) calls the special method `accept()` to immediately accept the input and `reject()` to immediately reject the input.
Equivalence of TMs and Programs

• Here's a sample program we might use to model a Turing machine for \( w \in \{a, b\}^* \mid w \) has the same number of \( a \)'s and \( b \)'s \):

```c
int main() {
    string input = getInput();
    int difference = 0;

    for (char ch: input) {
        if (ch == 'a') difference++;
        else if (ch == 'b') difference--;
        else reject();
    }

    if (difference == 0) accept();
    else reject();
}
```
Equivalence of TMs and Programs

• Now, a new fact: it’s possible to build a method `mySource()` into a program, which returns the source code of the program.

• For example, here's a narcissistic program:

```plaintext
int main() {
    string me = mySource();
    string input = getInput();

    if (input == me) accept();
    else reject();
}
```
Equivalence of TMs and Programs

- Sometimes, TMs use other TMs as subroutines.
- We can think of a decider for a language as a method that takes in some number of arguments and returns a boolean.
- For example, a decider for \{ a^n b^n \mid n \in \mathbb{N} \} might be represented in software as a method with this signature:
  
  ```
  bool isAnBn(string w);
  ```
- Similarly, a decider for \{ \langle m, n \rangle \mid m, n \in \mathbb{N} \text{ and } m \text{ is a multiple of } n \} might be represented in software as a method with this signature:
  
  ```
  bool isMultipleOf(int m, int n);
  ```
Side note: what does equivalence of TMs and programs mean for YOU?

- In this class (starting Problem Set Nine, and unless directed otherwise) you can write proofs about TMs by just writing normal code (e.g., Java or C++), and never have to painstakingly draw an actual TM ever again!

:: rejoicing ::
Self-Referential : Danger:

- So, I hope we’ve convinced you that there’s nothing magic, impossible, or scary about a program getting a string version of its own code.

- But, there are some dragons in the land of self-referential things....
True or false:

"This string is 34 characters long."
True or false:

"This string is 34 characters long."

1234567890123456789012345678901234
True or false:

"This sentence is written in blue."
In a certain isolated town, every house has a lawn and the city requires them all to be mowed. The town has only one gardener, who is also a resident of the town, and this gardener mows the lawns of every resident who does not mow their own lawn.
In a certain isolated town, every house has a lawn and the city requires them all to be mowed. The town has only one gardener, who is also a resident of the town, and this gardener mows the lawns of every resident who does not mow their own lawn.

**True or false:** The gardener mows their own lawn.
MY NOSE WILL GROW NOW!
True or false:

"This sentence is false."