Turing Machines
Describing Turing Machines

- Formal description

\[ M = \text{"On input string } w:\]
1. Sweep across tape, crossing off every other 0.
2. If tape contains one 0, accept.
3. Else, if number of 0’s is odd, reject.
4. Return head to left-hand end of tape.
5. Go to step 1.”

- High-level description
Recursive and Recursively Enumerable Languages

**Definition** (Turing decidable). A language is *Turing-decidable* if it there is a Turing machine that accepts every string in the language and rejects every string not in the language (thus the machine *always halts*). Turing decidable languages are also referred to as *recursive languages*.

If a Turing machine $M$ halts (accepts or rejects) for every input, we say that $M$ is a *decider*.

**Definition** (Turing recognizable). A language is *Turing-recognizable* if there is some Turing machine that *accepts* all the strings in the language. Turing recognizable languages are also referred to as *recursively enumerable languages*. 
Why Study Turing Machines?

- Turing machines are a terrible model for thinking about *fast* computation.

- We are not interested in finding fast algorithms, but rather in proving that some problems cannot be solved by any computational means.

- To do this, we require a formal definition of “computation” that is simple enough to support formal argument, but still powerful enough to describe *arbitrary algorithms*. 
Church-Turing Thesis (1936)

Intuitive Notion of Algorithms = Turing Machine Algorithms
Computability

Hilbert’s Tenth Problem (1900).

Devise an algorithm* that tests whether a given polynomial

\[ p(x_1, x_2, \ldots, x_n) \]

has an integral root.

*“find a process according to which it can be determined by a finite number of operations”
Languages and Problems

**Definition.** Let $D = \{ p \mid p \text{ is a polynomial with an integral root} \}$.

**Hilbert’s Tenth Problem.**

Determine whether $D$ is Turing-decidable.
Turing Machine (High-level) = Algorithmic Description

A high-level description of a Turing machine is a description of the form

\[ M = \text{“On input } x: \text{Do something with } x.” \]

What is allowed? Rule of thumb:

You can include anything in a high-level description, as long as you are convinced that, \textit{if you had to}, you could design a (low-level) Turing machine for it.
Recognizing \( L = \{a^n b^n c^n \mid n \geq 0\} \)

Exercise. Give the high-level description of a Turing machine that recognizes \( L \).

In the following we assume that tape symbols \( x, y, z \notin \Sigma \)

Let \( M = \) “On input string \( w \):

1. Scan the input from left to right to check if it is of the form \( a^*b^*c^* \) and reject if it is not.
2. Return head to left hand end of tape.
3. While there are \( a \)’s remaining on the tape, do:
   - Replace the first \( a \) with an \( x \), scan right until a \( b \) occurs; replace it with \( y \), and scan to the right until a \( c \) occurs; replace it with \( z \). If the corresponding \( b \) and \( c \) for each \( a \) are not found, reject.
4. If there are no \( b \)’s or \( c \)’s remaining on the tape, accept. Otherwise, reject.
Variants of Turing Machine

- Many equivalent ways to define a Turing Machine
  - To show two models are equivalent we need to show we can simulate each by the other

- Invariance to certain changes in model makes the TM a *robust* model

- Today we will discuss the following variants
  - Multitape Turing machines
  - Nondeterministic Turing machine
  - Enumerators
Formally, we need only change the transition function to \( \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k \).
Multitape TM $\iff$ Single-tape TM

- A multitape TM can always simulate a single-tape TM
- Can a single-tape TM simulate a multitape TM?
Multitape TM ⇔ Single-tape TM

- A multitape TM can always simulate a single-tape TM
- Can a single-tape TM simulate a multitape TM?

**Theorem.** Every multitape Turing machine has an equivalent single-tape Turing machine.
Nondeterministic TMs: Guess and Check

● One intuition for nondeterminism is perfect guessing.
  ○ The machine has many options, and somehow magically knows which guess to make.

● More formally, a NTM is a TM variant which can take any number of transitions for a given (state, tape symbol) combination \( \delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}) \)

● The computation of a NTM is a tree whose branches correspond to different possibilities for the machine. If some branch leads to an accept state, NTM accepts, that is, it accepts \textit{iff there exist some possible series of choices which make it accept.}
Designing NTMs

- When designing NTMs, it is often useful to use the approach of **guess and check**

- Nondeterministically guess some object that can “prove” that $w \in L$.

- Deterministically verify that you have guessed the right object.

- If $w \in L$, there will be some guess that causes the machine to accept.

- If $w \notin L$, then no guess will ever cause the machine to accept.
Theorem. Every nondeterministic TM has an equivalent deterministic TM.
Exercises

- Let a $k$-PDA be a pushdown automaton that has $k$ stacks.
  - Is a 1-PDA more powerful than a 0-PDA?
  - Is a 2-PDA more powerful than a 1-PDA?
  - Is a 3-PDA more powerful than a 2-PDA?