Last Time

• Exam

• Before that: Turing Machines!
  – A deterministic Pushdown Automaton where we can access anything on the stack – unlimited and unrestricted memory
  – If a problem is solvable, a Turing machine can solve it!
  – Operations: read/write to tape, move head left/right
Today

- Turing Machine Variants
  - Stay-Put Turing Machine
  - Multitape Turing Machine
  - Nondeterministic Turing Machine
  - Enumerator

- Why do we need variants?
  - Easier to represent some problems
  - Show they’re equivalent to regular TMs by proving that one can simulate the other
Stay-Put Turing Machine

• Instead of the transition function being \( \delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \), we will update it to \( \delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\} \).

• Is this machine equivalent to a regular Turing machine?
  – Yes: start with a vanilla Turing machine, and following every action, move in the opposite direction on any input.
Multitape Turing Machine

• It’s a Turing machine with more than one tape...
  – Each tape has its own read/write head
  – Input appears on tape 1; other tapes start out blank
  – New transition function:
    • \( \delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\} \), \( k \) is the number of tapes
    • \( \delta: (q_i, a_1, ..., a_k) = (q_j, b_1, ... b_k, L, ..., R) \)
Multitape Turing Machine

• Is this machine equivalent to a regular TM?
  – **Proof Idea:** Show how to simulate multitape machine $M$ with single-tape machine $S$.
    • $S$ needs to simulate $k$ tapes on a single tape.
      – Pick a new symbol (say, ‘#’) to delimit the different tapes.
      – Pick a new symbol (say, ‘∙’) to mark the location of each head.
Multitape Turing Machine

• **Proof:** $S = \text{“On input } w = w_1 w_2 \ldots w_n, \text{ do:”}~$

1. First, $S$ puts its tape into the format that represents all $k$ tapes of $M$. The formatted tape contains:

   
   
   
   

2. To simulate a single move, $S$ scans its tape from the left to right to determine the symbols under the virtual heads. Then, $S$ makes a second pass to update the tapes according to the way that $M'$s transition function dictates.

3. If at any point $S$ moves on of the virtual heads to the right onto a $\#$, $S$ writes a blank symbol on this tape cell, and shifts all tape contents to the right of that blank one unit to the right, then continues the simulation.”
Nondeterministic Turing Machine

• It’s a Turing machine that has multiple computational paths...
  – The computation is a tree whose branches correspond to different possibilities for the machine.
  – If some branch of the computation leads to the accept state, the machine accepts its input.
  – New transition function:
    • $\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$
Nondeterministic Turing Machine

• Is this machine equivalent to a regular TM?
  – **Proof idea:** Show how a deterministic TM $D$ can simulate all possible branches of nondeterministic TM $N$’s computation.
    • Root of the computational tree is the start configuration.
    • Each node of the tree is a configuration along some path of the computation.
    • $D$ shouldn’t do a DFS in case of an infinite computation, so let $D$ explore the tree using BFS.
    • $D$ will visit every node in the tree until it encounters an accepting configuration.
Nondeterministic Turing Machine

- Use three tapes: input tape, simulation tape, address tape
Proof: \( D = \text{"On input } w = w_1w_2 \ldots w_n, \text{ do:"} \) 

1. Initially, tape 1 contains the input \( w \), and tapes 2 \& 3 are empty. 
2. Copy tape 1 to tape 2 and initialize the string to tape 3 to be \( \varepsilon \). 
3. Use tape 2 to simulate \( N \) with input \( w \) on one branch of its nondeterministic computation. Before each step of \( N \), consult the next symbol on tape 3 to determine which choice to make among those allowed by \( N \)'s transition function. If no more symbols remain on tape 3, or if this ND choice is invalid, abort to step 4. Also, goto step 4 if a rejecting configuration is found. If an accepting configuration is found, accept. 
4. Replace the string on tape 3 with the next string in the string ordering. Simulate the next branch of \( N \)'s computation by going to step 2."
 Enumerator

• It’s a Turing machine with a printer attached...
  – The TM can use that printer as an output device to print strings. Every time the TM wants to add a string to the list, it sends the string to the printer.
  – Basically, an enumerator can print all strings in a language.
Enumerator

• **Theorem:** A language is Turing-recognizable iff some enumerator enumerates it.

• **Proof:** First, we show that if we have an enumerator $E$ that enumerates a language $A$, a TM $M$ recognizes $A$. The TM $M$ works like:
  
  $M = \text{“On input } w:\$
  1. Run $E$. Every time that $E$ outputs a string, compare it with $w$.
  2. If $w$ ever appears in the output of $E$, accept.”

• Next, we show the other direction: if a TM $M$ recognizes a language $A$, we can construct an enumerator for $A$:
  
  $E = \text{“Ignore all input.}$
  1. Repeat the following for $i = 1, 2, 3, \ldots$
  2. Run $M$ for $i$ steps on each input $s_1, s_2, \ldots, s_n$
  3. If any computations accept, print out the corresponding $s_j$.\hfill ■
A Programming Languages Analogue

• Languages like Perl and BASIC look completely different from each other.
• Is there an algorithm that you can write in Perl that you cannot write in BASIC?
Any Questions?

HOMEWORK (due 10/30)
3.9, 3.13, 3.15abc, 3.16abc