Last time: defined Turing machines [H-M-U Chapter 8.2]
programming techniques [H-M-U 8.3]
- subroutines
- multiple tracks for storing data on the tape
- storage in the finite state memory
  idea: think of a state as labeled by
(q, X, ..., X_k)
control
  state finite amount of memory
  unlike writing to the tape, this memory
  is always immediately accessible, like
  CPU registers.

Example: \( L = \{ x \in \{0, 1\}^* \mid x \leq 1 \} \)

Enhancements of Turing machines
1. Two-way infinite tape
2. Multiple heads
3. Multi-tape Turing machine
4. Nondeterministic Turing machine

Main claim: A standard Turing machine can simulate a
Turing machine with these extra features.

⇒ Justifies the Church-Turing thesis: a Turing machine is
the most general model of computation.
2-way tapes

\[
\begin{array}{c}
| \_ \_ \_ | \_ \_ \_ | \_ \_ \_ |
\end{array}
\]

Thm: Given 2-way TM \( M_2 \), a 1-way TM \( M_1 \) such that \( L(M_1) = L(M_2) \).

Proof sketch:

\( M_1 \) simulates \( M_2 \) using two tracks

\[
\begin{array}{cccccccc}
\# & B & B & B & \cdots & B \\
\# & w_1 & w_2 & w_3 & \cdots & w_n & b & \text{blanks}
\end{array}
\]

\( M_2 \) \hspace{1cm} \( M_1 \)

state

\( q \rightarrow (q, w), (q, L) \) upper lever

\( \Delta = \mathbb{Q} \times \Sigma \)

\( \Gamma' = \{ [x], 1x, y \in \Gamma \} \cup \{ \} \)

\( \Gamma' = \{ (q, w), (q, L) | q \in \mathbb{Q} \} \)

transitions

\( s'(q, w) = (q, b, R) \)

\( s'((q, L), [x]) = (q, L), [x] \) \( R \)

\( s'((q, w), [x]) = (q, w), [x] \) \( R \)

also \( \forall q \in \mathbb{Q} \) \( s((q, w), [x], [y]) = (q, w), [y] \) \( R \)

\( s((q, w), [x], [y]) = (q, w), [y] \) \( R \)

to switch tracks

Multihead T. M

\[
\begin{array}{c}
\text{finite state}
\end{array}
\]

independent heads w/ independent transitions

e.g. \( s((q, h_1, h_2, \ldots, h_k) = (p, (w_1, m_1), \ldots, (w_k, m_k)) \)

\( h_j \in \Gamma' \) head \( j \)'s cell

\( w_j \in \Gamma' \) what to write over \( h_j \)

\( m_j \in \Gamma' \) move \( h_j \)
Theorem: Given $k$-head $T_M$. $M_k$, and a 1-head $T_M$. $M_1$, s.t. $L(M_1) = L(M_k)$.

Idea:
Use $k+1$ tracks.

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>B</th>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>B</td>
<td>*</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>*</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>Tape</td>
<td>$w_1$</td>
<td>$w_2$</td>
<td>$w_3$</td>
<td></td>
</tr>
</tbody>
</table>

For each transition of $M_k$, make sequence of moves:
Start at leftmost cell

1. Sweep right: Until a * has been seen in each track, memorizing each by passing a *
2. Sweep left: Changing head positions and writing the *s following $M_k$'s rule
Note: you will sometimes have to jog right one step to move a * over

Multitape $T_M$:

Simulation:
Use 2k tracks

Simulation speed: These enhancements affect efficiency, if not expressive power.

Claim: If $M_k$ halts in $\leq T(n)$ steps on inputs of length $n$, then $M_1$ halts in $O(T(n)^3)$ steps.

Proof: Each simulated step requires sweeping entirely across the tape.

The number of non-empty cells on the tape is $\leq T(n)$. ✦
Nondeterministic Turing machines
allows multiple transitions
\[ f : Q \times \Sigma \rightarrow 2^{Q \times \Sigma \times \{L, R\}} \]
\[ (q, a) \rightarrow \{(p_1, b_1, D_1), ..., (p_r, b_r, D_r)\} \]

execution tree:
- (run forever)
- (crash)
- \(\bowtie\) accept!

machine accepts if any branch of execution leads to accepting state.

Theorem: Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof: Idea: Let \( N \) be a nondeterministic TM.
Construct \( D \) a deterministic TM, that simulates \( N \)
along all possible branches of \( N \)'s execution tree.
\( D \) accepts if it ever finds an accepting state.

Key point: Use breadth-first search, not depth-first search.
With breadth-first search, any node in an infinite tree will be explored.

Implementation: Use a 3-tape machine:
- \( D \)
- \( \text{input tape} \)
- \( \text{simulation tape} \)
- \( \text{execution tree address tape} \)

Let \( b = \max_{q \in Q} |\delta(q, x)| = \max \text{# of transition choices ever available} \)
\( \max \text{# of children of any node} \)

addresses \( \in \{1, 2, ..., b\}^* \)
\( \epsilon = \text{root} \)
\( 3, 2, 1 = \text{accepting state} \)
\( 2, 1 = \text{invalid state} \)

1. Initially tape 1 has input \( w \), tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2.
3. Use tape 2 to simulate \( N \) along the computation branch determined
by tape 3. Next symbol on tape 3 gives choice to make. If choice is invalid,
\( N \) rejects, or no symbols remain, go to 4. If \( N \) accepts, then accept.
4. Increment tape 3 to lex.-next string. Clear tape 2. Go to 3.
**Simulation Efficiency**

Def. Halting TM = TM that accepts or rejects (by getting stuck) any input so within a finite # of moves.

Def. NTM N has running time T(n) if it halts in 
\[ \leq T(n) \] transitions, on any input of length n, for all possible seq. of transitions.
(Also applies to DTM.)

Def. Polynomial time if \( T(n) \leq n^c \) for some fixed \( c \).

eg. \( n^{1000} \), \( n^{\log n} \), \( 2^n \times 1.00001^n \times \)

Def. Efficient TM = any NTM with poly-running time.

Earlier simulations preserved efficiency.

but for simulating an NTM with time \( O(T(n) \cdot r(n)) \) not polynomial.

- In worst case must try all possible transition sequences.

Open problem: Can efficient NTMs be efficiently simulated? P = NP?

**Decidability Theory**

Def: Algorithm = Halting T.M.

Def: Procedure = TM (which may not halt).

(Always halts in finite time for all inputs)

- Yes
- No

Recursive lang. = accepted by some alg.

Undecidable problem = language for which no alg. exists.

Observe: No difference between NTM & DTM for these defas.

\( L \text{ is r.e. } \Leftrightarrow \exists \text{ TM that output everything in some order.} \)