Topics Covered

- Turing machines
- Variants of Turing machines
  - Multi-tape
  - Non-deterministic
- Definition of algorithm
- The Church-Turing Thesis
Finite State Automata

- Can be simplified as follow

- State control for states and transitions
- Tape to store the input string
Pushdown Automata

- Introduce a stack component

- Symbols can be read and written there
Turing Machine (TM)

- Introduce an infinite tape

- Symbols can be read and written there
- Move left and right on the tape
- Machine accepts, rejects, or loops
Turing Machine (TM)

- Let’s design one for the language
  \[ F = \{ w\#w \mid w \in \{0,1\}^* \} \]

- How will it work?

- Remember:
  - It has the string on the tape
  - It can go left and right
  - It can write symbols on the tape
Turing Machine (TM)

\[ F = \{ w\#w \mid w \in \{0,1\}^* \} \]

The machine does this:
- Scan to check there is only one #
- Zig-zag across # and read symbols
- If do not match reject
- If they match write the symbol x
- If all symbols left to # match, accept
Turing Machine (TM)

\[ F = \{ w\#w \mid w \in \{0,1\}^* \} \]

\( w_1 \in F = "011000\#011000" \)
Formal Definition of a TM

A Turing machine is a 7-tuple

\( (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}) \)

- \( Q \) is the set of states
- \( \Sigma \) is the input alphabet, without \( \sqcup \)
- \( \Gamma \) is the tape alphabet and \( \sqcup \in \Gamma, \Sigma \subseteq \Gamma \)
- \( \delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \) is the transition function
- \( q_0 \in Q \) is the initial state
- \( q_{\text{accept}}, q_{\text{reject}} \in Q \) are the final states
TM Configurations

- Describe the state of the machine
- Written as $C = uq_i v$ where:
  - $q_i$ is the current state of the machine
  - $uv$ is the content of the tape
  - The head stays at the first symbol of $v$
**TM Transitions**

- A configuration $C_1$ yields $C_2$ if the machine can go from $C_1$ to $C_2$ in 1 step

- $uaq_i bv$ yields $uq_j acv$ if $\delta(q_i, b) = (q_j, c, L)$

- $uaq_i bv$ yields $uacq_j v$ if $\delta(q_i, b) = (q_j, c, R)$

- Note: cannot go over the left border!
TM Acceptance

- The machine starts at $q_0 w$
- The machine accepts at $q_{\text{accept}}$
- The machine rejects at $q_{\text{reject}}$

- An input is accepted if there is $C_1, \ldots, C_k$
  - The machine starts at $C_1$
  - Each $C_i$ yields $C_{i+1}$
  - $C_k$ is an accepting state
Computations and Deciders

- Three possible outcomes:
  - It ends in an accept state
  - It ends in a reject state
  - It does not end (loops forever)

- Accept and reject are halting states
- Loops are not halting
- A **Decider** halts on every input
TMs and Languages

- The strings a TM $M$ accepts define the language of $M$, $L(M)$

- A language is Turing recognizable (recursively enumerable) if some TM recognizes it

- A language is Turing decidable (recursive) if some TM decides it
TM Example

TM $M_2$ recognizes the language consisting of all strings of zeros with their length being a power of 2. In other words, it decides the language

$$A = \{0^{2^n} \mid n \geq 0\}.$$
1. Sweep left to right across the tape, crossing off every other 0
2. If the tape has a single 0, accept
3. If the tape has more than one 0 and the number of 0s is odd, reject
4. Return the head to the left
5. Go to stage 1

\[ A = \{0^{2^n} \mid n \geq 0\} \]
TM Example

The image shows a transition diagram for a Turing Machine. The states are labeled as follows:

- $q_1$
- $q_2$
- $q_3$
- $q_4$
- $q_5$
- $q_{\text{accept}}$
- $q_{\text{reject}}$

The transitions are labeled with input symbols and actions:

- $0 \rightarrow □, R$
- $x \rightarrow R$
- $0 \rightarrow □, R$
- $0 \rightarrow x, R$
- $0 \rightarrow □, L$
- $x \rightarrow □, R$
- $0 \rightarrow □, L$
- $0 \rightarrow x, R$
- $x \rightarrow □, R$
- $0 \rightarrow □, R$
- $x \rightarrow □, R$

The machine starts in state $q_1$ and ends in state $q_{\text{accept}}$ or $q_{\text{reject}}$ depending on the accepted or rejected input.
Another TM Example

\[ F = \{w#w \mid w \in \{0,1\}\} \]

1. Check for \#, if not reject
2. Zig-zag across and cross off same symbols. If not same, reject
3. If all left of \# are crossed, check for non crossed symbols on the right side
4. If none, accept, otherwise reject
Another TM Example
Variants of Turing Machines

- Mostly equivalent to the original

- Example: consider movements as \{L,R,S\}, where S means stay still

- Equivalent to original, represent S as two transitions: first R, then L or vice versa
Multi-Tape Turing Machine

- Include multiple tapes and heads
- Input on first tape, the others blank
- Transitions $\delta: Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$
Equivalence Result

Theorem 3.13:
Every multitape Turing machine has an equivalent single-tape Turing machine.
Equivalence Result

**Theorem 3.13:**
Every multitape Turing machine has an equivalent single-tape Turing machine.
Theorem 3.13:
Every multitape Turing machine has an equivalent single-tape Turing machine.
Proof of Theorem 3.13

- Consider a input $w_1 w_2 \ldots w_k$
- Add dotted symbols for the head
- Put all the input on the single tape
  \[
  \# w_1 w_2 \ldots w_k \# \# \# \# \ldots \#
  \]
- Simulate a single move
  - Scan from first $\#$ to last to get the heads
  - Re-run to update the tape
- If head symbols go to the right $\#$ write a blank and shift the tape content
Equivalence Result

Corollary 3.15:
A language is Turing-recognizable if and only if some multi-tape Turing machine recognizes it

Proof:
Forward: an ordinary machine is a special case of a multi-tape
Backward: see Theorem 3.13
**Intermezzo: Programming**

“Brainfuck”: language *simulating* a TM

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>increment the data pointer (to point to the next cell to the right). <strong>R</strong></td>
</tr>
<tr>
<td>&lt;</td>
<td>decrement the data pointer (to point to the next cell to the left). <strong>L</strong></td>
</tr>
<tr>
<td>+</td>
<td>increment (increase by one) the byte at the data pointer.</td>
</tr>
<tr>
<td>-</td>
<td>decrement (decrease by one) the byte at the data pointer.</td>
</tr>
<tr>
<td>.</td>
<td>output a character, the ASCII value of which being the byte at the data pointer.</td>
</tr>
<tr>
<td>,</td>
<td>accept one byte of input, storing its value in the byte at the data pointer.</td>
</tr>
<tr>
<td>[</td>
<td>if the byte at the data pointer is zero, then instead of moving the <em>instruction pointer</em> forward to the next command, <em>jump</em> it forward to the command after the matching ] command.</td>
</tr>
<tr>
<td>]</td>
<td>if the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump it back to the command after the matching ] command*.</td>
</tr>
</tbody>
</table>

*(http://en.wikipedia.org/wiki/Brainfuck)*
Non Deterministic TMs (NTMs)

- Transition function changed into
\[ \delta: Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\}) \]
\[ \delta(q, a) = \{(q_1, b_1, L), \ldots, (q_k, b_k, R)\} \]

- Same idea as for NFAs
Theorem 3.16:
Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Idea:
- Three tapes: input, simulation, index
- Simulator to perform computation
- Index to trace the path in the tree
Theorem 3.16:
Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

- Idea:
  - Three tapes: input, simulation, index
  - Simulator to perform computation
  - Index to trace the path in the tree
Proof of Theorem 3.16

1. Copy the input from tape 1 to 2
2. Use tape 2 to simulate N on one branch of computation
   a. Consult tape 3 to get the transition
   b. Abort if empty symbol, invalid or reject
   c. Accept if accept state
3. Replace the string on 3 with the lexicographically next one
4. Repeat from 1.
NTMs and Languages

Corollary 3.18:
A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.

Corollary 3.19:
A language is decidable if and only if some nondeterministic Turing machine decides it.
Enumerators

- Recursively enumerable languages
- Recognized by TMs
- Alternative model: Enumerator

![Diagram of state control and work tape]
Enumerators

- *Enumerate* the strings
- Start with empty tape
- Output tape (printer)
- Print strings instead of accepting them
- Printing in any order
- Strings might be duplicated


**Equivalence Result**

**Theorem 3.21:**
A language is Turing-recognizable if and only if some enumerators enumerate it.

**Proof:**

*Forward:* e have an enumerator E. We can build a machine T that

1. Run E and compare every string
2. If it appears, accept
Equivalence Result

Backward: We have a machine T. We can build an enumerator E as this:

1. Ignore the input
2. For each i = 1, 2, ...
   1. Run T for i steps on each input in Σ*
   2. If any computation accepts, print it.

E eventually prints all string T accepts
Other Variants of TMs

- Many other variants of TMs exist
- All equivalent in power under reasonable assumptions
- *Turing complete* languages
- The class of algorithms described identical for all these languages.
- For a given task, one type of language may be more elegant or *simple*.
Definition of Algorithm

- Precise definition only in 20th century
- Informal idea was already present
- Collection of instructions for a task
- Formal definition needed to be found
Anecdote: David Hilbert

- Famous mathematician
- Int. congress of Maths in 1900
- Formulated 23 math problems

- The 10th problem said:
  - Devise an algorithm to test whether a polynomial has an integral root
  - Algorithm = “a process according to which it can be determined by a finite number of operations”
Anecdote: David Hilbert

- Mathematicians believed it existed
- We know it is not possible
- A formal definition of algorithm was needed to prove it
- Alonso Church: $\lambda$-calculus
- Alan Turing: Turing machines
- Church—Turing Thesis:
  - Intuitive algorithm = TM algorithm
Formal Definition of Algorithm

- Let’s rephrase Hilbert problem
- Consider the set
  \[ D = \{ p \mid p \text{ is a polynomial with integer root} \} \]
- Hilbert problem asks if D is decidable
- Unfortunately it is not
- Fortunately is Turing recognizable
Formal Definition of Algorithm

- Consider a simpler problem
  
  \[ D_1 = \{ p \mid p \text{ is a poly. over } x \text{ with integer root} \} \]

- Build a TM that recognizes it
  
  1. Input is a polynomial over \( x \)
  2. Evaluate \( p \) with \( x=0,1,-1,2,-2,... \)
  3. If polynomial evaluates to 0, accept
Formal Definition of Algorithm

- Describe an algorithm equals to describe a Turing machine
- Three possibilities:
  - Formal description (low level)
  - Implementation description (mid level)
  - *English* description (high level)

- We will describe machines in high level
Turing Machine Description

- Input is always a string
- Objects represented as strings
- Encoding is irrelevant (equivalence)
- TM Algorithm will be high level
- First line describe the input
- Indentations describe blocks
Example description

\[ A = \{ \langle G \rangle \mid G \text{ is a connected undirected graph} \} \]

Remember the definition of connected?
Example description

\[ A = \{\langle G \rangle \mid G \text{ is a connected undirected graph} \} \]

Remember the definition of connected?
Every node is reachable from every one

\[ G = \]

![Graph diagram](attachment:graph.png)
M = “On input <G>, the encoding of a graph G:
1. Select the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked.
   1. For each node in G, mark it if it is attached by an edge to a node that is already marked.
3. Scan all the nodes of G to determine whether they all are marked.
   If yes, accept; otherwise reject."
Summary

- Turing machines
- Variants of Turing machines
  - Multi-tape
  - Non-deterministic
- The definition of algorithm
  - The Church-Turing Thesis