Lecture One

- Finite automata have limited memory.
- Pushdown automata, memory only used as last-in-first-out stack.

- Turing Machines proposed by Alan Turing 1936.
  Main differences:
  1) A Turing machine can both write on the tape and read from it.
  2) The read-write head can move both left & right.
  3) The tape is infinite.
  4) The special states for rejecting and accepting take effect immediately.

Formal Definition: A Turing machine is a 7-tuple
\( (Q, \Sigma, \Gamma, \delta, q_0, q_\text{accept}, q_\text{reject}) \), where \( Q, \Sigma, \Gamma \) are all finite sets and:
1. \( Q \) is the set of states.
2. \( \Sigma \) is the input alphabet (not containing the blank symbol \( \lambda \)).
3. \( \Gamma \) is the tape alphabet, where \( \Sigma \subseteq \Gamma \subseteq \{\lambda\} \).
4. \( \delta \): \( Q \times \Gamma \rightarrow Q \times \{L, R\} \) is the transition function.
5. \( q_0 \in Q \) is the start state.
6. \( q_\text{accept} \in Q \) is the accept state.
7. \( q_\text{reject} \in Q \) is the reject state, \( q_\text{accept} \neq q_\text{reject} \).

- Turing machine \( M \) receives input \( w_0 w_1 w_2 \ldots w_n \in \Sigma^* \) on the leftmost n squares of the tape. The rest of the tape is blank. (i.e., filled with \( \lambda \)).
- The head starts on the leftmost square of the tape.
- The first \( \lambda \) marks the end of input.

- \( \vdots \)
- Computation proceeds according to the rules of the transition function.
- If \( M \) ever turns to move head right of the left end, the tape remains at the same place, even though the transition function indicates \( L \).
- The computation halts in accepting or rejecting status. If neither happens, \( M \) goes on forever.

Configuration of the Turing machine: \( q_0 \).

1) Current tape content is \( u \). \( V \).
2) Current state is \( q_1 \).
3) Current head location is \( \# \) for symbol \( u \).
4) Tape contains only blank symbols after the last symbol \( \# \).

\[ \begin{array}{c}
1 & 0 & 9 & 5 & 1 & 0 & 1 \\
\end{array} \]

- \( a \) \( q_0 \) \( b \) \( a \) \( c \) \( a \) \( c \) \( u \) \( v \) \( w \) \( \ldots \)

- \( u a q_0 b v \) yields \( U a q_0 c a w \) if \( \delta(q_0, b) = (q_1, a, L) \)
- \( u a q_0 b v \) yields \( U a q_0 c a v \) if \( \delta(q_0, b) = (q_1, c, R) \)
- \( q_0 b v \) yields \( q_0 v \) if \( \delta(q_0, b) = (q_0, c, L) \)
- \( U a q_0 b v \) yields \( U a q_0 c w \) if \( \delta(q_0, b) = (q_0, c, R) \)

- Start configuration \( q_0 b v \).
- Accepting configuration the state is \( q_0 \) and \( \{ \# \} \) holding.
- Rejecting configuration the state is \( q_0 \) and \( \{ \# \} \) not holding.

"M accepts \( w \)" if a sequence of configurations \( \psi \in \ldots \psi_k \psi_0 \) exists where:
1. \( \psi_0 \) is the start configuration \( q_0 b v \).
2. Each \( \psi_i \) yields \( \psi_{i+1} \).
3. \( \psi_k \) is an accepting configuration.

Define language recognized by \( M \) as the language of \( \psi \), denoted \( L(M) \) as the collection of strings accepted by \( M \).
Def: (Turing recognizable): Language L is Turing recognizable if some Turing machine recognizes it.

A.k.a. Recursively Enumerable Language:

Def: (Turing): A Turing machine that halts on all inputs (new loops).
A Turing machine computes L is said to decide L.

Def: (Turing-decidable): Language L is Turing-decidable or simply decidable if some Turing machine decides it.
A.k.a. Recursive Language.

Example: A: \{ \epsilon^n | n \geq 0 \}.

Implementation level description:

\[ M = \langle Q, \Sigma, \Gamma, q_0, \delta, F \rangle \]

1. Scan left to right across the tape, moving off any other 0.
2. If in state 1, the tape contains a single 0, accept.
3. If in state 2, the tape contains more than a single 0 and the number of 0s are odd, reject.
4. Return the head to the left hand end of the tape.
5. Go to step 1.

Formal Description:

\[ Q = \{ q_0, q_1, q_2, q_3, q_4 \} \]
\[ \Sigma = \{ 0 \} \]
\[ \Gamma = \{ 0, x, L \} \]
\[ q_0 \quad \text{start state} \]

\[ \delta \]

\[ 1) \quad q_0 \rightarrow q_2 \rightarrow q_1 x \rightarrow q_3 \rightarrow q_4 \rightarrow q_4 \]
\[ 2) \quad q_0 \rightarrow q_1 \rightarrow q_0 \rightarrow q_0 \rightarrow q_3 \rightarrow q_1 x \rightarrow q_2 \rightarrow q_3 \rightarrow q_3 \]
\[ 3) \quad q_0 \rightarrow q_1 \rightarrow q_0 \rightarrow q_0 \rightarrow q_3 \rightarrow q_3 \rightarrow q_3 \]

Variants of Turing Machine:
Variants of Turing Machines:

1. "Many-in One-out" $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R,S\}$
   It is equivalent.

2. Multitape Turing Machine: $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R,S\}^k$
   where $k$ is the number of tapes
   $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$

Theorem: Every multitape machine has an equivalent single tape machine.

Proof:

- Start at $S$ and mark the first cell of the input tape as $\#$.

2. To simulate a single move of multitape machine $M$, $S$ makes a first pass from left to right and updates the tape according to $M$.

3. If at any point $S$ wants to reach cell $i$ on a $\#$, the
   It writes blank symbol $0$.

Corollary: A language is Turing recognizable if and only if some multitape Turing machine recognizes it.

3. Non-Deterministic Turing Machine:

Theorem: Every non-deterministic Turing Machine $N$ has an equivalent deterministic Turing machine $D$.

Proof:

- Initially tape contains input $w$, tape $2$ and $3$ are empty.

2. Copy tape $2$ to $tpe$.

3. If, then $2$ simulates $N$ with input $w$ on the branch of
3. Use type 2 to simulate N with input s on the branch of
   non-deterministic computation given by type 3. If no more symbols
   are left on type 3 or the address is invalid, goto Stage 4.
   If a rejecting configuration is encountered, goto Stage 4.
   If an accepting configuration is encountered, accept the input.

4. Replace string on type 3 with lexicographically next string. Goto Stage 2.

Corollary: A language in Turing recognizable iff some non-deterministic Turing machine
recognizes it.

- We can modify the construction so that if N halts on all branches of
  its computation D halts.
- Call a non-deterministic Turing machine "deciding" if all branches halt
  on all inputs.

Corollary: A language is decidable iff some non-deterministic Turing machine decides it.

\[ \text{Enumerator} \]

[Diagram: Enumerator, Enum starts with blank input tape.
- It may not halt and may print infinite list of strings.
- The language enumerated by E is the collection of all strings eventually printed by it.
- E may generate the strings of language in any order possibly with repetition.

This: A language in Turing-recognizable iff some Enumerator enumerates it.

Proof: (\( \Rightarrow \)), let E be enumerator for language A. Turing machine M recognizing A:

M: \( \text{On input } w: \)
1. Run E. Every time E prints a string, compare it with w. If it
   matches, accept.
2. If no match appears, accept.

(\( \Rightarrow \)) Let M recognize A. Enumerator for A:

E: \( \text{Ignore the input.} \)
1. Repeat the following for \( i = 1, 2, 3, \ldots \) steps:
   a. Run M for \( i \) steps on each input \( s_1, s_2, \ldots, s_i \).
   b. If any computations accept, print out the corresponding \( s_j \)
Church–Turing Thesis:

- Informally speaking an algorithm is a collection of simple instructions for carrying out some task.

  \[\text{Intuitive notion of Algorithm} \iff \text{Turing machine algorithm}\]

The Church–Turing Thesis:

- Alonzo Church in 1936 used a notation system called the λ-calculus to define algorithms.
- Turing did it with his machines, also in 1936.
- The two definitions were shown to be equivalent.
- Many other definitions are also made later but are all shown equivalent.
- Church–Turing thesis states that "any reasonable definition is equivalent."

"Reasonable definition" for example requiring that any finite amount of work be done in single step.

Hilbert's Tenth Problem:

- In 1900, David Hilbert proposed 23 problems for the 20th century.
  - At International Congress of Mathematicians in Paris.
  - The tenth problem was to find "a process according to which it can be determined by a finite number of operations if a given polynomial has an integral root."
  - Hilbert assumed that such an algorithm existed.

- In 1970, Yuri Matijasevič, building on the work of Martin Davis, Hilary Putnam, and Julia Robinson, showed that No algorithm exists to do this task.
- For some varieties polynomials, no algorithm can be given.

Describing Turing Machines:

1) Formal description.

2) Implementation description: The way Turing machine head moves and the way it stores data on the tape etc. Ignore details of states or transition function.

3) High-level description: Describe the algorithm, ignoring implementation details (Default unless otherwise specified).
• For an object \( D \), its string representation \( <0> \).

• For multiple objects \( D_1, D_2, \ldots, D_k \), single string \( <D_1, D_2, \ldots, D_k> \).

• Any encoding is fine.

• If the input \( <D> \), the Turing machine first implicitly tests whether the input properly encodes an object of the desired form and rejects otherwise. This step need not be written in the algorithm explicitly.

Example: \( A = \{<G> | G \text{ is a connected undirected graph} \} \)

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 unless no new nodes are marked:
   a. 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 check if they are all marked.
   If they are accept, otherwise reject."