## Chapter 4: Turing Machines

A. Basics

1. Motivation.
   - What is the "next most powerful" model of a computation?
   - Adding a stack to a FA was helpful, but a more general model of memory seems necessary.
   - How can we model computations that don't correspond to language recognition (e.g., function evaluations)?
   - Note: we don't care about efficiency at all (yet).
   - Preview: the next model (Turing machines) will be the last one we need (probably).

2. **Definition**. A **Turing Machine** (TM) is a quadruple $M = (K, \Sigma, \delta, s)$ where

   $K$ is a finite set of states,
   $\Sigma$ is an alphabet,
   $s \in K$ is the
   $\delta : K \times \Sigma \to (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$ is a

3. Terminology and notation.
   - A TM $M$ consists of a finite control and a one-way infinite tape.
   - At each step $M$

   - $\delta(q, a) = (p, b)$ means

   - The special blank symbol is $\# \in \Sigma$; movement is indicated by $L, R \notin \Sigma$.

- Termination? There are three possibilities:

- Conventions:
  - The input to a TM begins in the second square from the left.
  - Initially there is a # in the leftmost square and an infinite number of #'s to the right of the input.
  - The read/write head is initially over the first # to the right of the input.
  - On halting the head should be over the first # to the right of the output.
- A **configuration** of a machine $M$ is a member of

  We generally write $(q, w\underline{a}u)$ rather than $(q, w, a, u)$.
- One configuration **yields** another:

4. Examples.

5. Turing-computable functions.
   a. **Definition**. A function $f$ is **Turing-computable** if there exists a TM $M$ such that

$$f(w) = u \quad \Longrightarrow$$

   b. Example.
   c. Generalizations:
      - A function from $\Sigma_0^* \times \Sigma_0^* \times \cdots \times \Sigma_0^*$ to $\Sigma_1^*$:
      - A function from $\mathcal{N}$ to $\mathcal{N}$:
      - A function from $\mathcal{N}^k$ to $\mathcal{N}$.
      - Example.

6. Turing-decidable languages.
   a. **Definition**. A language $L \subseteq \Sigma_0^*$ is **Turing-decidable** if the function $\chi_L : \Sigma_0^* \rightarrow \{\mathbf{Y}, \mathbf{N}\}$ is Turing-computable, where

$$\chi_L(w) =$$

   b. Example.

7. Turing-acceptable languages.
   a. **Definition**. A language $L \subseteq \Sigma_0^*$ is **Turing-acceptable** if there exists a TM $M$ such that
   b. Example.
   c. Note: not every Turing-acceptable language is Turing-decidable!

8. Combining Turing machines.
   a. Idea: build up more complicated TMs by combining simple TMs; we can view simple TMs as subroutines.
   b. Simplest machines:
      - "Symbol-writing machine":
      - "Head-moving machine":
      - Some more useful machines:

c. Definitions:

- A **machine schema** is a triple $(\mathcal{M}, \eta, M_0)$ where

  $\mathcal{M}$ is a finite set of TMs with a common alphabet $\Sigma$ and disjoint state sets,
  $M_0 \in \mathcal{M}$ is the initial machine,
  $\eta$ is a function from a subset of $\mathcal{M} \times \Sigma$ to $\mathcal{M}$.
- The computation (TM) represented by a machine schema.
  Let $\mathcal{M} = \{M_0, \ldots, M_m\}$ where $M_i = (K_i, \Sigma, \delta_i, s_i)$.
  Let $q_0, \ldots, q_m$ be new states not in any $K_i$.
  Then the machine schema $(\mathcal{M}, \eta, M_0)$ represents the TM $M = (K, \Sigma, \delta, s)$ where
  $K = K_0 \cup \cdots \cup K_m \cup \{q_0, \ldots, q_m\}$,
  $s = s_0$,
  and $\delta$ is defined by:
  If $\delta_i(q, a) = (p, b)$ and $p \neq h$ then $\delta(q, a) = (p, b)$.
  If $\delta_i(q, a) = (h, b)$ then $\delta(q, a) = (q_i, b)$.
  If $\eta(M_i, a)$ is undefined then $\delta(q_i, a) = (h, a)$.
  If $\eta(M_i, a) = M_j$ and $\delta_j(s_j, a) = (p, b)$ then

$$\delta(q_i, a) = \begin{cases} (p, b) & \text{if } p \neq h, \\ (q_j, b) & \text{if } p = h. \end{cases}$$

d. Example.

e. A graphical representation for machine schema.

f. Examples.

B. Extensions

1. Two-way infinite tape
   - Idea: allow tape to be infinite in both directions.
   - Example:
   - **Theorem 4.5.1** (paraphrase). Anything you can do with a two-way infinite tape TM you can do with a standard TM.
   - Idea of the simulation. "Fold" two-way infinite tape in half; the symbols on the one-way tape are ordered pairs of symbols from the two-way machine; the state keeps track of which half the machine is "really" looking at.

2. Multiple tapes
   - Idea: allow $k$ one-way tapes; at each step machine reads one symbol from each tape and does one thing (write or move) on each tape.
   - Example:
   - **Theorem 4.5.2** (paraphrase). Anything you can do with a $k$-tape TM you can do with a standard TM.
   - Idea of the simulation. Simulate $2k$ "tracks" on one tape using $2k$-tuple symbols; use 2 tracks for each tape from the $k$-tape machine: one for tape contents and one for head position.

3. Other ideas:

4. Nondeterminism
   - Recall: nondeterminism made no real difference for FAs, but it did for PDAs.
   - Idea: allow a choice of what to do next, given a current state and symbol.
   - Formally,

   - Notes:
     - We will focus on NDTMs as acceptors since nondeterminism doesn't really help with computing functions or deciding languages.
     - A NDTM $M$ accepts $w$ if for some computation, $M$ halts on input $w$.
   - Example.
   - **Theorem 4.6.1.** For every NDTM $M_1$ there is a standard TM $M_2$ such that for every string $w$, $M_1$ halts on $w$ iff $M_2$ halts on $w$.
   - Idea of the proof:
     - $M_2$ tries all possible $M_1$ computations on a given input in a systematic way: first all one-step computations, then all two-step computations, etc.
     - If $M_1$ halts for some sequence of steps, then $M_2$ will too.
     - It helps to describe $M_2$ as a 3-tape machine (see text).