Turing Machines and General Computability

The *Turing Machine* is a simple model which, apparently, totally captures mechanical computability. TM's are much simpler than Java programs, so they are easier to understand, yet they capture the power of Java programs.

First proposed by the British mathematician Alan Turing in the 1936, at a time when mathematicians were trying to develop a notion of effective computation. Several formal systems were developed.

Formal models and the Church-Turing thesis

- Turing machines (TM's) (Alan Turing)
- Post systems (Emil Post)
- μ-recursive functions (Kurt Godel, Jacques Herbrand)
- λ-calculus (Alonzo Church, Stephen C. Kleene)
- combinatory logic (Moses Schonfinkel, Haskell B. Curry) And later on,
- Game of life (Conway)
- Neural nets

All these systems can simulate one another and are computationally equivalent.

Church-Turing Thesis: Every effective computation (or algorithm) can be described by a Turing machine; the Turing machine exactly captures our notion of effective computation.

Turing Machines

A Turing machine contains:

1. A *finite set of states* including two final states, one *accept* and one *reject* state.
2. A *tape* divided into *cells*, each of which can hold any one of a finite number of symbols. The tape is infinite (in one direction.)
3. A *read-write head* always positioned at one of the tape cells which can move right or left..

Operation of a Turing Machine

Initially,

- the *input* which is a finite-length string of symbols is placed one in each cell starting with the leftmost end of the tape; all cells not containing input contain the *blank* symbol $\sqcup$.
- the tape head is at the leftmost cell that holds the input.

In a single step, depending on its current state and the tape symbol currently scanned, the Turing machine can:

1. Change state,
2. Replace the symbol currently scanned by another symbol AND
3. Move the tape head one tape square to the left or right.
4. If there are no more cells to move left to and the machine is supposed to move left, it stays in the same place.

Formal Definition of a TM

A Turing machine is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$where

$$
\begin{aligned}
Q &= \{\text{states}\} \\
\Sigma &= \text{set of input symbols not including } \sqcup \\
\Gamma &= \text{set of tape symbols, including input symbols and } \sqcup \\
\delta &= \text{ transition function where } \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \\
q_0 &= \text{start state} \\
q_{accept} \in Q &= \text{accept state} \\
q_{reject} \in Q &= \text{reject state, } q_{reject} \neq q_{accept}
\end{aligned}
$$

Transition Function

We define for $q \in Q, X \in \Gamma$, $\delta(q, X) = (p, Y, D)$, where

- p is the next state;
- $Y \in \Gamma$ is written in the cell being scanned, replacing the current symbol in the cell;
- D is a *direction* of the move, either L or R.

Computation on a TM

An *configuration* of a TM is a string $X_1X_2...X_{i-1}qX_iX_{i+1}...X_n$ where

- q is the state of the TM;
- The tape head is scanning the i^{th} symbol from the left;
- $X_1X_2...X_n$ is the portion of the tape up to the rightmost nonblank. An exception is if the head is on a blank.
- We say configuration A yields configuration B if you can get to B from A with one application of δ
- *start configuration*
- *halting configurations*

TM's as language recognizers

- A string w is *accepted* by a TM M if it reaches q_{accept} when it is started with w written on the input tape, with the read head on the leftmost tape square (what else could happen?)

- The set of strings accepted by a TM M is denoted $L(M)$ and is called the *language recognized by M*.

- A language we can accept using a TM is called *Turing-recognizable* or *recursively enumerable* or *RE*.

Halting

- A TM *halts* if it reaches a configuration $X_1X_2...X_{i-1}qX_iX_{i+1}...X_n$ and $\delta(q, X_i)$ is not defined.

- A TM is assumed to halt once it enters an accepting or rejecting state (so we assume, for $q = q_{accept}, q_{reject}$ and any X, $\delta(q, X)$ is always undefined.)

- It is not always possible to design a TM which halts even when it doesn't accept a string (Turing)

- If L is recognized by some M which halts on every input, it is called *Turing-decidable*, *decidable* or *recursive*.

Example 1

$L = \{a^n b^n : n \geq 0\}$.

Idea: Replace a with a' then go right until b is encountered and change b to a'. Go back and repeat. If there aren't enough b's, die; if the a's are gone, check if the b's are gone. If so, accept.

Let $M = (\{q_0, q_1, q_2, q_3, q_a, q_r\}, \{a, b\}, \{a, b, a', b', \sqcup\}, \delta, q_0, q_a, q_r)$,

	$\underline{a}$	$\underline{b}$	$\underline{a'}$	$\underline{b'}$	$\underline{\sqcup}$
q_0	(q_1, a', R)	(q_r, a', R)	(q_r, a', R)	(q_3, b', R)	
q_1	(q_1, a, R)	(q_2, b', L)	(q_r, a', R)	(q_1, b', R)	
q_2	(q_2, a, L)	(q_r, a', R)	(q_0, a', R)	(q_2, b', L)	
q_3	(q_r, a', R)	(q_r, a', R)	(q_r, a', R)	(q_3, b', R)	$(q_a, \sqcup, R)$

Example 2

$L = \{ww | w \in \{a, b\}^*\}$.

- Idea: First, mark the symbol which starts the second w, then compare each symbol of the first w with its corresponding symbol in the second w.

- How to do this?