6.1 The Turing Machine

**Definition** A *deterministic Turing machine* (DTM) $M$ is specified by a sextuple $(Q, \Sigma, \Gamma, \delta, s, f)$, where

- $Q$ is a finite set of *states*;
- $\Sigma$ is an alphabet of *input* symbols;
- $\Gamma$ is an alphabet of *tape* symbols, where $(\Sigma \cup \{B\}) \subseteq \Gamma$;
- $\delta$ is a *transition function* $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, \lambda\}$, where $L$, $R$, and $\lambda$ denote move left, move right, and do not move, respectively;
- $s$ in $Q$ is a *start state*; and
- $f$ in $Q$ is a *final state*. 
A DTM is initialized by

1. resetting the read-write head over the first cell of the tape;
2. letting the state of the DTM be $s$;
3. placing an input word $x$ over $\Sigma$ into the first $|x|$ cells, one symbol to a cell; and
4. assuming $B$ is in all other cells.
We capture all information about the DTM’s current situation with the notion of a *configuration*, as we did with FAs. A configuration is a word in $\Gamma^* Q \Gamma^*$, which denotes the current state, the position of the read-write head, and also the currently written portion of the tape. In a configuration $g_1 pg_2$, the read-write head is assumed to be over the first symbol of $g_2$ if $g_2 \neq \lambda$ and over a blank cell otherwise. Strictly speaking, $g_2$ is either $\lambda$ or a word in $\Gamma^*(\Gamma - \{B\})$; there are no trailing blanks. If $\Gamma$ and $Q$ have elements in common, it is possible to have configuration that contain more than one state. Such a configuration does not identify a unique situation in the DTM, so we forbid this by assuming $\Gamma \cap Q = \emptyset$, throughout.
In any configuration $g_1pg_2$, $M$ proceeds in one of three ways:

1. If $p = f$, then $M$ halts and, by definition, no move is possible. This is the reason we only require one final state.

2. If $g_2 = \lambda$, then the read-write head is over a blank cell. In this case, if $\delta(p, B) = (q, X, D)$, then $M$ enters state $q$, rewriting the blank cell with $X$ and moving one cell to the left or right or remaining where it is, depending on $D$.

3. If $g_2 = ag_3$, where $a$ is in $\Gamma$, $g_3$ is in $\Gamma^*$, and $\delta(p, a) = (q, X, D)$, then $a$ is rewritten as $X$, $M$ enters state $q$, and the read-write head moves according to $D$.

4. In (2) and (3) if $\delta(p, B)$ or $\delta(p, a)$ is undefined, then $M$ hands, that is, no move is possible.
Definition Let \( M = (Q, \Sigma, \Gamma, \delta, s, f) \) be a DTM. Then for two configurations \( gph \) and \( g'p'h' \) we write \( gph \vdash g'p'h' \) if

1. either \( h = Ah_1 \), for some \( A \) in \( \Gamma \) and \( h_1 \) in \( \Gamma^* \), or \( h = \lambda \) and by convention \( A = B \) and \( h_1 = \lambda \);
2. \( \delta(p, A) \) is defined, and \( p \neq f \);
3. \( \delta(p, A) = (p', B, D) \) and one of the following holds
   (a) \( D = L \), then \( g = g'C \) for some \( C \) in \( \Gamma \), and \( h' = CBh_1 \);
   (b) \( D = \lambda \), then \( g' = g \) and \( h' = Bh_1 \);
   (c) \( D = R \), than \( g' = gB \) and \( h' = h_1 \).
Notice that condition (3)(a) implies that \( g \neq \lambda \), since a move left is impossible in this case; in other words, \( M \) hangs once more.

We also write \( gph \vdash^i g'p'h' \), for some \( i > 0 \), if either \( i = 1 \) and \( gph \vdash g'p'h' \) or \( i > 1 \), \( gph \vdash g''p''h'' \). Form some \( g''h'' \) in \( \Gamma^* \) and \( p'' \) in \( Q \), and \( g''p''h'' \vdash^{i-1} g'p'h' \).

We write \( gph \vdash^+ g'p'h' \) if \( gph \vdash^i g'p'h' \) for some \( i > 0 \) and we write \( gph \vdash^* g'p'h' \) if either \( gph = g'p'h' \) or \( gph \vdash^+ g'p'h' \).
In an FA the finiteness of the input together with the absence of rewriting ensures that the reading head will eventually fall of the tape to the right. However, it is possible that a DTM may neither hang nor halt.

**Definition** Let $M = (Q, \Sigma, \Gamma, \delta, s, f)$ be a DTM. A word $x$ in $\Sigma^*$ is *accepted* by $M$ if $M$ terminates in a configuration $gfh$, when given $x$ as input, that is,

$$sx \vdash^* gfh$$

The corresponding configuration sequence is called an *accepting configuration sequence*. Otherwise, either $x$ causes $M$ to hang or $M$ never terminates.
We say two DTMs, $M_1$ and $M_2$, are equivalent if $L(M_1) = L(M_2)$. The family of languages specified by all DTMs is denoted by $L_{DTM}$ and defined by

$$L_{DTM} = \{L : L = L(M) \text{ for some DTM } M\}.$$  

A language in $L_{DTM}$ is said to be a deterministic Turing machine language or a DTML.
Since DTMs were introduced as a general model for computation it is worthwhile seeing how we might program them to carry out some basic operations, rather than using them only as language recognizers.

**Definition** Let $M = (Q, \Sigma, \Gamma, \delta, s, f)$ be a DTM. Then $M$ computes the function $f_M : \Sigma^* \rightarrow (\Gamma - \{B\})^*$ that is defined by

- For all $x$ in $\Sigma^*$,

  $$f_M(x) = y \text{ in } (\Gamma - \{B\})^* \text{ iff } sx \vdash^* y_1fy_2, \text{ where } y = y_1y_2$$
Definition Let y and n be tape symbols representing yes and no. Then a DTM $M = (Q, \Sigma, \Gamma, \delta, s, f)$ is said to be a decision-making Turing machine if y and n are in $\Gamma$ and not in $\Sigma$, and

- For all $x$ in $\Sigma^*$, either $sx \vdash^* fy$ or $sx \vdash^* fn$ in $M$.

The yes language of $M$, denoted by $Y(M)$, is defined as

$$Y(M) = \{ x : x \text{ is in } \Sigma^* \text{ and } sx \vdash^* fy \}$$

and the no language of $M$, denoted by $N(M)$, is defined as

$$N(M) = \{ x : x \text{ is in } \Sigma^* \text{ and } sx \vdash^* fn \}$$
Such a DTM is a decision maker since for each word in $\Sigma^*$ it always halts and gives either a yes or a no answer. Note that $L(M) = Y(M) \cup N(M) = \Sigma^*$ and $Y(M) \cap N(M) = \emptyset$.

**Definition** Given an alphabet $\Sigma$ and a language $L \subseteq \Sigma^*$ we say that $L$ is *decidable* if there is a decision-making Turing machine $M = (Q, \Sigma, \Gamma, \delta, s, f)$ with $Y(M) = L$.

The languages that are decidable by decision-making Turing machines form an important family – the family of recursive languages. This is denoted by $L_{REC}$ and is defined by

$$L_{REC} = \{Y(M) : M \text{ is a decision-making Turing machine}\}$$
6.2 Turing Machine Programming