Turing machines

Criteria:

- They should be automata
- They should be as simple as possible to describe
- They should be as general as possible

The tape has a left end, but it extends indefinitely to the right.

LP pg 180

Formal definition:

A **Turing machine** is a quintuple $(K, \Sigma, \delta, s, H)$ where

- K is a finite set of states
- Σ is an alphabet, including $\sqcup$ (blank) and $\triangleright$ (left-end-of-tape), but not $\leftarrow$ or $\rightarrow$.
- $s \in K$ is the initial state
- $H \subseteq K$ is the set of halting states
- δ is the transition function from $(K - H) \times \Sigma$ to $K \times (\Sigma \cup \{\leftarrow, \rightarrow\})$
- For all $q \in K - H$, if $\delta(q, \triangleright) = (p, b)$, then $b = \rightarrow$
- For all $q \in K - H$ and $a \in \Sigma$, if $\delta(q, a) = (p, b)$, then $b \neq \triangleright$

Ex 4.1.1: $K = \{q_0, q_1, h\}, \Sigma = \{\mathtt{a}, \sqcup, \triangleright\}, s = q_0\}, H = \{h\}$

q	σ	$\delta(q, \sigma)$
q_0	$\mathtt{a}$	$(q_1, \sqcup)$
q_0	$\sqcup$	$(h, \sqcup)$
q_0	$\triangleright$	$(q_0, \rightarrow)$
q_1	$\mathtt{a}$	$(q_0, \mathtt{a})$
q_1	$\sqcup$	$(q_0, \rightarrow)$
q_1	$\triangleright$	$(q_1, \rightarrow)$

LP pg 182

Ex 4.1.1: $K = \{q_0, h\}, \Sigma = \{\mathtt{a}, \sqcup, \triangleright\}, s = q_0, H = \{h\}\}$

q	σ	$\delta(q, \sigma)$
q_0	$\mathtt{a}$	$(q_0, \leftarrow)$
q_0	$\sqcup$	$(h, \sqcup)$
q_0	$\triangleright$	$(q_0, \rightarrow)$

LP pg 183

Prob 4.1.1: $K = \{q_0, q_1, h\}, \Sigma = \{\mathtt{a}, \mathtt{b}, \sqcup, \triangleright\}, s = q_0\}, H = \{h\}$

q	σ	$\delta(q, \sigma)$
q_0	$\mathtt{a}$	$(q_1, \mathtt{b})$
q_0	$\mathtt{b}$	$(q_1, \mathtt{a})$
q_0	$\sqcup$	$(h, \sqcup)$
q_0	$\triangleright$	$(q_0, \rightarrow)$
q_1	$\mathtt{a}$	$(q_0, \rightarrow)$
q_1	$\mathtt{b}$	$(q_0, \rightarrow)$
q_1	$\sqcup$	$(q_0, \rightarrow)$
q_1	$\triangleright$	$(q_1, \rightarrow)$

LP pg 191

Definition 4.1.2: Configuration:

$$K \times \triangleright\Sigma* \times (\Sigma*(\Sigma - \{\sqcup\}) \cup \{\varepsilon\})$$

Definition 4.1.3: $\vdash_M$ means *transition in one step* to a new state *and* either write, go left, or go right.

Definition 4.1.4:

- One configuration **yields** another: $C_0 \vdash_M^* C_2$
- A **computation** is a sequence of configurations
- A computation has **length** n or n **steps**, $C_0 \vdash_M^n C_n$.

$$
\begin{tikzcd}[column sep=large]
> L_{\sqcup} \arrow[r] & R \arrow[r, "a \neq \sqcup"] \arrow[d, "\sqcup"] & \sqcup R^2_{\sqcup} a L^2_{\sqcup} a \arrow[l, bend right] \\
& R_{\sqcup} &
\end{tikzcd}
$$

LP pg 190. Figure 4-8, redrawn

$$
\begin{tikzcd}
> L \arrow[rr, "a \neq \sqcup"] \arrow[d, "\sqcup"] & & \sqcup R_{\sqcup} a L_{\sqcup} \arrow[ll, bend right] \\
R_{\sqcup}
\end{tikzcd}
$$

LP pg 190. Figure 4-9, redrawn and corrected

Definition as language acceptor:

A **Turing machine** is a quintuple $(K, \Sigma, \delta, s, H)$

- $H = \{y, n\}$
- M **accepts** w if $(s, \triangleright\underline{\sqcup}w) \vdash^*_M (y, x)$
- M **rejects** w if $(s, \triangleright\underline{\sqcup}w) \vdash^*_M (n, x)$
- M **decides** language $L \subseteq \Sigma_0^*$ if $\forall\ w \in \Sigma_0^*$, if $w \in L$, then M accepts w; and if $w \notin L$, then M rejects w.
- A language L is **recursive** if there exists a Turing machine that decides L.

  The term "recursive," as a synonym for "decidable," goes back to mathematics as it existed prior to computers. Then, formalisms for computation based on recursion (but not iteration or loops) were commonly used as a notion of computation. These notations. . . had some of the flavor of computation in functional programming languages such as LISP or ML. In that sense, to say a problem was "recursive" had the positive sense of "it is sufficiently simple that I can write a recursive function to solve it, and the function always finishes." That is exactly the meaning carried by the term today, in connection with Turing machines.

  Hopcroft et al, Automata Theory, Languages, and Computation, *pg 385.*

say that M decides a language $L \subseteq \Sigma_0^*$ if for any string $w \in \Sigma_0^*$ the following is true: If $w \in L$ then M accepts w; and if $w \notin L$ then M rejects w.

Finally, call a language L **recursive** if there is a Turing machine that decides it.

That is, a Turing machine decides a language L if, when started with input w, it always halts, and does so in a halt state that is the correct response to the input: y if $w \in L$, n if $w \notin L$. Notice that no guarantees are given about what happens if the input to the machine contains blanks or the left end symbol.

Example 4.2.1: Consider the language $L = \{a^n b^n c^n : n \geq 0\}$, which has heretofore evaded all types of language recognizers. The Turing machine whose diagram is shown in Figure 4-11 decides L. In this diagram we have also utilized two new basic machines, useful for deciding languages: Machine y makes the new state to be the accepting state y, while machine n moves the state to n.

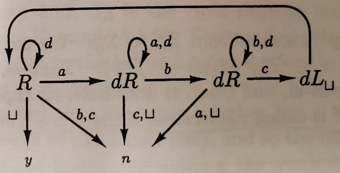

Figure 4-11

Definition 4.2.4: Let $M = (K, \Sigma, \delta, s, H)$ be a turing machine, $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$ be an alphabet and $L \subseteq \Sigma_0^*$ be a language.

- M **semidecides** L if

$$\forall\ w \in \Sigma_0^*, w \in L \text{ iff } M \text{ halts on } w$$

- L is **recursively enumerable** iff there exists a Turing machine that semidecides L.