\( M = (Q, \Sigma, \delta, s) \)

- \( Q \): finite set of states; \( s \in Q \)
- \( \Sigma \): finite set of symbols; \( \triangleright, \sqcup \in \Sigma \)
- \( \delta \): \( Q \times \Sigma \rightarrow (Q \cup \{h\}) \times \Sigma \times \{←, →, −\} \)

\[
\begin{array}{c|cccccc}
\text{s} & \triangleright & 1 & 1 & 0 & 1 & \sqcup \\
\end{array}
\]
<table>
<thead>
<tr>
<th>mvRt.tm</th>
<th>s</th>
<th>q</th>
<th>q₀</th>
<th>q₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s, 0, →</td>
<td>q₀, □, →</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>s, 1, →</td>
<td>q₁, □, →</td>
<td></td>
<td></td>
</tr>
<tr>
<td>□</td>
<td>q, □, ←</td>
<td>s, 0, ←</td>
<td>s, 1, ←</td>
<td></td>
</tr>
<tr>
<td>▶</td>
<td>s, ▶, →</td>
<td>h, ▶, −</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**comment**

- find □
- memorize & erase
- change □ to 0
- change □ to 1

---

```
s ▶[1]101□□...

s ▶[1]101□□...

s ▶[1]1□01□□...

s ▶11[0]1□□...

s ▶110[1]□□...

s ▶1101□□...

q ▶110[1]□□...

q₁ ▶110□□□□...

s ▶110□[1]1□...
```
<table>
<thead>
<tr>
<th>mvRt.tm</th>
<th>( s )</th>
<th>( q )</th>
<th>( q_0 )</th>
<th>( q_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( s,0,\rightarrow )</td>
<td>( q_0,\square,\rightarrow )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( s,1,\rightarrow )</td>
<td>( q_1,\square,\rightarrow )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \square )</td>
<td>( q,\square,\leftarrow )</td>
<td>( s,0,\leftarrow )</td>
<td>( s,1,\leftarrow )</td>
<td></td>
</tr>
<tr>
<td>( \triangleright )</td>
<td>( s,\triangleright,\rightarrow )</td>
<td>( h,\triangleright,\leftarrow )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{c}
\hline
s & \square & 1 & 1 & 0 & 1 & \square & \square & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
q_1 & \square & 1 & 1 & 0 & \square & \square & \square & \cdots \\
s & \square & 1 & 1 & 0 & \square & 1 & \square & \cdots \\
q & \square & 1 & 1 & 0 & \square & 1 & \square & \cdots \\
q_0 & \square & 1 & 1 & \square & \square & 1 & \square & \cdots \\
s & \square & 1 & 1 & \square & 0 & 1 & \square & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
q & \square & 1 & 1 & 0 & 1 & \square & \cdots \\
h & \square & 1 & 1 & 0 & 1 & \square & \cdots \\
\hline
\end{array}
\]
Hilbert’s Program [1901]: Give a complete axiomization of all of mathematics!

Such a complete axiomization would have provided a mechanical procedure to churn out exactly all true statements in mathematics.

This led to active interest in 1930’s in the question: “What is a mechanical procedure?”

Church: Lambda calculus
Gödel: Recursive function
Kleene: Formal system
Markov: Markov algorithm
Post: Post machine
Turing: Turing machine

**Fact:** The above models are all define exactly the same class of “computable” functions.

**Church-Turing Thesis:** The intuitive idea of “effectively computable” is captured by the precise definition of “computable” in any of the above models.
“Why is a Turing machine as powerful as any other computational model?”

Intuitive answer: Imagine any computational device. It has:

- Finitely many states
- Ability to scan limited amount per step: one page at a time
- Ability to print limited amount per step: one page at a time
- Next state determined by current state and page currently being read (but what about randomization?)

**Note:** Without the potentially infinite supply of tape cells, paper, extra disks, extra tapes, etc. we have just a (potentially huge) **finite state machine**.

The PC on your desk, with 20 GB of hard disk is a finite state machine with over $2^{160,000,000,000}$ states!

This is better modeled as a TM with a bounded number of states, and an “infinite tape”, actually meaning a finite memory that expands whenever necessary.
\[
M(w) \equiv \begin{cases} 
y & \text{if } M \text{ on input } \downarrow w \uparrow \text{ eventually} \\
\downarrow & \text{halts with output } \downarrow y \downarrow 
\end{cases}
\]

\[
\Sigma_0 \equiv \Sigma - \{\downarrow, \uparrow\}
\]

Usually, \(\Sigma_0 = \{0, 1\}\)

**Definition 4.1** Let \(f : \Sigma^* \rightarrow \Sigma^*_0\) be a total or partial function. We say that \(f\) is **recursive** iff \(\exists \ TM \ M, \ f = M(\cdot)\), i.e.,

\[
(\forall w \in \Sigma^*_0) \ f(w) = M(w).
\]
Remark 4.2 There is an easy to compute 1:1 and onto map between \( \{0, 1\}^* \) and \( \mathbb{N} \). Thus we can think of the contents of a TM tape as a natural number and talk about \( f : \mathbb{N} \to \mathbb{N} \) being recursive. (We may visit this issue in HW#2.)

Partial function \( f : \mathbb{N} \to \mathbb{N} \) is a total function \( f : D \to \mathbb{N} \) where \( D \subseteq \mathbb{N} \). A partial function that is not total is called strictly partial. If \( n \in \mathbb{N} - D, f(n) = \uparrow \).
Definition 4.3 Let $S \subseteq \Sigma_0^*$ or $S \subseteq \mathbb{N}$.

$S$ is a recursive set iff the function $\chi_S$ is a (total) recursive function,

$$\chi_S(x) = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{otherwise} \end{cases}$$

$S$ is a recursively enumerable set ($S$ is r.e.) iff the function $p_S$ is a (partial) recursive function,

$$p_S(x) = \begin{cases} 1 & \text{if } x \in S \\ \text{otherwise} \end{cases}$$

Proposition 4.4 If $S$ is recursive then $S$ is r.e.

Proof: Suppose $S$ is recursive and let $M$ be the TM computing $\chi_S$.

Build $M'$ simulating $M$ but diverging if $M(x) = 0$. Thus $M'$ computes $p_S$.  

\[\spadesuit\]
Proposition 4.5 The following functions are recursive. They are all total except for \( \text{even} \).

\[
\begin{align*}
\text{copy}(w) &= \text{ww} \\
\sigma(n) &= n + 1 \\
\text{plus}(n, m) &= n + m \\
\text{times}(n, m) &= n \times m \\
\text{exp}(n, m) &= n^m \\
\chi_{\text{even}}(n) &= \begin{cases} 
1 & \text{if } n \text { is even} \\
0 & \text{otherwise}
\end{cases} \\
\text{peven}(n) &= \begin{cases} 
1 & \text{if } n \text { is even} \\
& \text{otherwise}
\end{cases}
\end{align*}
\]

Proof: Exercise: please convince yourself that you can build TMs to compute all of these functions!
If $\mathcal{C}$ is any class of sets, define co-$\mathcal{C}$ to be the class of sets whose complements are in $\mathcal{C}$,

$$
\text{co-}\mathcal{C} = \{ S \mid \overline{S} \in \mathcal{C} \}
$$

**Theorem 4.6** $S$ is recursive iff $S$ and $\overline{S}$ are both r.e.

**Thus,** \textbf{Recursive} $= \text{r.e.} \cap \text{co-r.e.}$

**Proof:** If $S \in \text{Recursive}$ then $\chi_S$ is a recursive function.

Thus so is $\chi_{\overline{S}}(x) = 1 - \chi_S(x)$

Thus, $S$ and $\overline{S}$ are both recursive and thus both r.e.
Suppose $S \in \text{r.e.} \cap \text{co-r.e.}$

$p_S = M(\cdot); \quad p_{\overline{S}} = M'(\cdot)$

Define $T = M \parallel M'$ on input $x$:

1. \textbf{for} $n := 1$ to $\infty$ \{  
2. \quad run $M(x)$ for $n$ steps.  
3. \quad \textbf{if} $M(x) = 1$ in $n$ steps \textbf{then return}$(1)$  
4. \quad run $M'(x)$ for $n$ steps.  
5. \quad \textbf{if} $M'(x) = 1$ in $n$ steps \textbf{then return}$(0)$\}

Thus, $T(\cdot) = \chi_S$ and thus $S \in \text{Recursive.}$
**Arithmetic Hierarchy**

- co-r.e. complete
- co-r.e.
- r.e.
- r.e. complete

**Recursive**

**Primitive Recursive**

**EXPTIME**

**PSPACE**

**Polynomial-Time Hierarchy**

- co-NP complete
- co-NP
- NP
- NP ∩ co-NP

**P**

"truly feasible"

**NC**

**NC^2**

log(CFL)

SAC^1

**NSPACE[log n]**

**DSPACE[log n]**

**Regular**

**Logarithmic-Time Hierarchy**

**AC^0**

**ThC^0**

**NC^1**

**NSPACE**