Turing machines
Algorithms, informally

No precise definition of “algorithm” at the time Hilbert posed the *Entscheidungsproblem*, just examples.

Common features of the examples:

- finite description of the procedure in terms of *elementary operations*
- deterministic (next step uniquely determined if there is one)
- procedure may not terminate on some input data, but we can recognize when it does terminate and what the result is.

*E.g.* multiply two decimal digits by looking up their product in a table.
Register Machine computation abstracts away from any particular, concrete representation of numbers (e.g. as bit strings) and the associated elementary operations of increment/decrement/zero-test.

Turing’s original model of computation (now called a Turing machine) is more concrete: even numbers have to be represented in terms of a fixed finite alphabet of symbols and increment/decrement/zero-test programmed in terms of more elementary symbol-manipulating operations.

TM control structure is also more elementary than for RMS
Turing machines, informally

machine is in one of a finite set of states

Linear tape, unbounded to right, divided into cells containing a symbol from a finite alphabet of tape symbols. Only finitely many cells contain non-blank symbols.
Turing machines, informally

Machine is in one of a finite set of states.

Linear tape, unbounded to right, divided into cells containing a symbol from a finite alphabet of tape symbols. Only finitely many cells contain non-blank symbols.
Turing machines, informally

- Machine is in one of a finite set of states.
- Tape symbol being scanned by tape head.
- Special left endmarker symbol.
- Special blank symbol.
- Linear tape, unbounded to right, divided into cells containing a symbol from a finite alphabet of tape symbols. Only finitely many cells contain non-blank symbols.
Turing machines, informally

- Machine starts with tape head pointing to the special left endmarker ▷.
Turing machines, informally

- Machine starts with tape head pointing to the special left endmarker $\triangleright$.
- Machine computes in discrete steps, each of which depends only on current state ($q$) and symbol being scanned by tape head (0).
Turing machines, informally

- Machine starts with tape head pointing to the special left endmarker ▷.
- Machine computes in discrete steps, each of which depends only on current state ($q$) and symbol being scanned by tape head (0).
- Action at each step is to overwrite the current tape cell with a symbol, move left or right one cell, or stay stationary, and change state.
Turing Machines

are specified by:

- \( Q \), finite set of machine states
- \( \Sigma \), finite set of tape symbols (disjoint from \( Q \)) containing distinguished symbols \( \triangleright \) (left endmarker) and \( \sqcup \) (blank)
- \( s \in Q \), an initial state
- \( \delta \in (Q \times \Sigma) \rightarrow (Q \cup \{\text{acc, rej}\}) \times \Sigma \times \{L, R, S\} \), a transition function—specifies for each state and symbol a next state (or accept acc or reject rej), a symbol to overwrite the current symbol, and a direction for the tape head to move (\( L \)=left, \( R \)=right, \( S \)=stationary).
Turing Machines

are specified by:

- \( Q \), finite set of machine states
- \( \Sigma \), finite set of tape symbols (disjoint from \( Q \)) containing distinguished symbols \( \triangleright \) (left endmarker) and \( \square \) (blank)
- \( s \in Q \), an \textit{initial state}
- \( \delta \in (Q \times \Sigma) \rightarrow (Q \cup \{\text{acc, rej}\}) \times \Sigma \times \{L, R, S\} \), a transition function, satisfying:

for all \( q \in Q \), there exists \( q' \in Q \cup \{\text{acc, rej}\} \) with \( \delta(q, \triangleright) = (q', \triangleright, R) \)

(i.e. left endmarker is never overwritten and machine always moves to the right when scanning it)
Example Turing Machine

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

where

states \( Q = \{s, q, q'\} \) \((s \text{ initial})\)

symbols \( \Sigma = \{\triangleright, \sqcup, 0, 1\}\)

transition function

\[ \delta \in (Q \times \Sigma) \rightarrow (Q \cup \{ \text{acc, rej} \}) \times \Sigma \times \{L, R, S\}: \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \triangleright )</th>
<th>( \sqcup )</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>(\text{rej, } \triangleright, R)</td>
<td>(q', \text{acc, } R)</td>
<td>(\text{rej, } 0, S)</td>
<td>(\text{rej, } 1, S)</td>
</tr>
<tr>
<td>q</td>
<td>(\text{rej, } \triangleright, R)</td>
<td>(q', 0, L)</td>
<td>(q, 1, R)</td>
<td>(q, 1, R)</td>
</tr>
<tr>
<td>q'</td>
<td>(\text{rej, } \triangleright, R)</td>
<td>(\text{acc, } \sqcup, S)</td>
<td>(\text{rej, } 0, S)</td>
<td>(q', 1, L)</td>
</tr>
</tbody>
</table>
Turing machine configuration: \((q, w, u)\)

where

- \(q \in Q \cup \{\text{acc}, \text{rej}\}\) = current state

- \(w\) = non-empty string \((w = va)\) of tape symbols under and to the left of tape head, whose last element \((a)\) is contents of cell under tape head

- \(u\) = (possibly empty) string of tape symbols to the right of tape head (up to some point beyond which all symbols are \(\_\))

(So \(wu \in \Sigma^*\) represents the current tape contents.)

\(\uparrow\) (not uniquely)
Turing machine computation: \((q, w, u)\)

where

- \(q \in Q \cup \{\text{acc}, \text{rej}\} = \) current state
- \(w = \) non-empty string \((w = va)\) of tape symbols under and to the left of tape head, whose last element \((a)\) is contents of cell under tape head
- \(u = \) (possibly empty) string of tape symbols to the right of tape head (up to some point beyond which all symbols are \(\square\))

Initial configurations: \((s, \triangleright, u)\)
Turing machine computation

Given a TM $M = (Q, \Sigma, s, \delta)$, we write

$$(q, w, u) \rightarrow_M (q', w', u')$$

to mean $q \not\in \text{acc, rej}$, $w = va$ (for some $v$, $a$) and

either $\delta(q, a) = (q', a', L)$, $w' = v$, and $u' = a'u$

or $\delta(q, a) = (q', a', S)$, $w' = va'$ and $u' = u$

or $\delta(q, a) = (q', a', R)$, $u = a''u''$ is non-empty, $w' = va'a''$ and $u' = u''$

or $\delta(q, a) = (q', a', R)$, $u = \varepsilon$ is empty, $w' = va' \sqcup$ and $u' = \varepsilon$. 
\[ \delta(q, a) = (q', a', L) \]

\[
\begin{array}{c}
q \\
\downarrow \\
v | a | u
\end{array}
\quad \overset{\sim}{\longrightarrow}
\quad
\begin{array}{c}
q' \\
\downarrow \\
v | a' | u
\end{array}
\]

\[
(q, va, u) \quad \rightarrow \quad \text{M} \quad (q', v, a'u)
\]
\[ \delta(q, a) = (q', a', S) \]

\[ (q, va, u) \rightarrow_M (q', va', u) \]
$\delta(q, a) = (q', a', R)$

$(q, v a, u) \xrightarrow{M} (q', ?, ?)$
\[ \delta(q, a) = (q', a', R) \]

\[
\begin{array}{c|c|c}
q & v & u \\
\hline
q' & v & u
\end{array}
\]

\[
(q, va, u) \rightarrow_m (q', ?, ?)
\]

Two cases:

\[
\begin{cases}
  u = a''u'' & \text{is non-empty} \\
  u = \varepsilon & \text{is empty}
\end{cases}
\]
\[ \delta(q, a) = (q', a', R) \]

\[
\begin{array}{c|c|c}
q & a & a''u'' \\
\hline
\end{array} \quad \sim \quad \begin{array}{c|c|c}
q' & va' & a''u'' \\
\hline
\end{array}
\]

\[(q, va, a''u'') \rightarrow_{M} (q', va'a'', u') \]

Two cases:

1. \( u = a''u'' \) is non-empty
2. \( u = \varepsilon \) is empty
\[ \delta(q, a) = (q', a', R) \]

\[
\begin{array}{c}
\downarrow \\
q
\end{array}
\quad \overset{\leadsto}{\rightarrow}
\quad
\begin{array}{c}
\downarrow \\
q'
\end{array}
\]

\[
(q, va, \varepsilon) \quad \rightarrow \quad (q', va', \varepsilon)
\]

Two cases:

\[
\begin{cases}
    u = a^n u' & \text{is non-empty} \\
    u = \varepsilon & \text{is empty}
\end{cases}
\]
Turing machine computation

A computation of a TM $M$ is a (finite or infinite) sequence of configurations $c_0, c_1, c_2, \ldots$

where

- $c_0 = (s, \triangleright, u)$ is an initial configuration
- $c_i \rightarrow_M c_{i+1}$ holds for each $i = 0, 1, \ldots$.

The computation

- does not halt if the sequence is infinite
- halts if the sequence is finite and its last element is of the form $(\text{acc}, w, u)$ or $(\text{ rej}, w, u)$. 
Example Turing Machine

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

where \( Q = \{s, q, q'\} \) (\( s \) initial)

symbols \( \Sigma = \{\rhd, \sqcup, 0, 1\} \)

transition function

\[ \delta \in (Q \times \Sigma) \rightarrow (Q \cup \{\text{acc, rej}\}) \times \Sigma \times \{L, R, S\} : \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \rhd )</th>
<th>( \sqcup )</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s )</td>
<td>( (s, \rhd, R) )</td>
<td>( (q, \sqcup, R) )</td>
<td>( \text{ rej,0,S} )</td>
<td>( \text{ rej,1,S} )</td>
</tr>
<tr>
<td>( q )</td>
<td>( \text{ rej,\rhd,R} )</td>
<td>( (q', 0, L) )</td>
<td>( (q, 1, R) )</td>
<td>( (q, 1, R) )</td>
</tr>
<tr>
<td>( q' )</td>
<td>( \text{ rej,\rhd,R} )</td>
<td>( (\text{ acc,\sqcup,S} ) )</td>
<td>( \text{ rej,0,S} )</td>
<td>( (q', 1, L) )</td>
</tr>
</tbody>
</table>

Claim: the computation of \( M \) starting from configuration \( (s, \rhd, \sqcup 1^n 0) \) halts in configuration \( (\text{acc, \rhd, } 1^{n+1} 0) \).
Example Turing Machine

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

states $Q = \{s, q, q'\}$ ($s$ initial)

symbols $\Sigma = \{\triangleright, \sqcup, 0, 1\}$

transition function

$\delta \in (Q \times \Sigma) \rightarrow (Q \cup \{\text{acc, rej}\}) \times \Sigma \times \{L, R, S\}$:

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\triangleright$</th>
<th>$\sqcup$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>$(s, \triangleright, R)$</td>
<td>$(q, \sqcup, R)$</td>
<td>$(\text{rej}, 0, S)$</td>
<td>$(\text{rej}, 1, S)$</td>
</tr>
<tr>
<td>$q$</td>
<td>$(\text{rej}, \triangleright, R)$</td>
<td>$(q', 0, L)$</td>
<td>$(q, 1, R)$</td>
<td>$(q, 1, R)$</td>
</tr>
<tr>
<td>$q'$</td>
<td>$(\text{rej}, \triangleright, R)$</td>
<td>$(\text{acc}, \sqcup, S)$</td>
<td>$(\text{rej}, 0, S)$</td>
<td>$(q', 1, L)$</td>
</tr>
</tbody>
</table>

Claim: the computation of $M$ starting from configuration $(s, \triangleright, \sqcup 1^n 0)$ halts in configuration $(\text{acc}, \triangleright, \sqcup 1^{n+1} 0)$. 

a string of $n$ 1s
The computation of \( M \) starting from configuration \((s, \triangleright, \sqcap 1^n0)\):

\[
\begin{align*}
(s, \triangleright, \sqcap 1^n0) &\rightarrow_M (s, \triangleright\sqcap, 1^n0) \\
&\rightarrow_M (q, \triangleright\sqcap 1^n0, 1^{n-1}0) \\
&\vdots \\
&\rightarrow_M (q, \triangleright\sqcap 1^n0, 0) \\
&\rightarrow_M (q, \triangleright\sqcap 1^{n+1}0, \varepsilon) \\
&\rightarrow_M (q', \triangleright\sqcap 1^{n+1}0, 0) \\
&\vdots \\
&\rightarrow_M (q', \triangleright\sqcap, 1^{n+1}0) \\
&\rightarrow_M (\text{acc}, \triangleright\sqcap, 1^{n+1}0)
\end{align*}
\]
The computation of $M$ starting from configuration $(s, \triangleright, \square 1^n0)$:

$$(s, \triangleright, \square 1^n0) \rightarrow_M (s, \triangleright\square, 1^n0)$$

$$(q, \triangleright\square 1^n, 0)$$

$$(q, \triangleright\square 1^n0, \varepsilon)$$

$$(q, \triangleright\square 1^{n+1}, 0)$$

$$(q', \triangleright\square 1^{n+1}0)$$

$$(\text{acc}, \triangleright\square, 1^{n+1}0)$$

\{tape head moving right\}

\{tape head moving left\}
Theorem. The computation of a Turing machine $M$ can be implemented by a register machine.

Proof (sketch).

Step 1: fix a numerical encoding of $M$’s states, tape symbols, tape contents and configurations.

Step 2: implement $M$’s transition function (finite table) using RM instructions on codes.

Step 3: implement a RM program to repeatedly carry out $\rightarrow_M$. 
Step 1

- Identify states and tape symbols with particular numbers:

  \[
  \begin{align*}
  \text{acc} &= 0 \\
  \text{rej} &= 1 \\
  Q &= \{2, 3, \ldots, n\} \\
  \downarrow &= 0 \\
  \uparrow &= 1 \\
  \Sigma &= \{0, 1, \ldots, m\}
  \end{align*}
\]

N.B.

- Code configurations \( c = (q, w, u) \) by:

  \[
  c \Downarrow = [q, \lfloor a_n, \ldots, a_1 \rfloor, \lfloor b_1, \ldots, b_m \rfloor]
  \]

  where \( w = a_1 \cdots a_n \) (\( n > 0 \)) and \( u = b_1 \cdots b_m \) (\( m \geq 0 \)) say.
Step 1

- Code configurations \( c = (q, w, u) \) by:

\[
\overline{c} = \overline{[q, \overline{[a_n, \ldots, a_1]}, \overline{[b_1, \ldots, b_m]}]} \]

where \( w = a_1 \cdots a_n \) (\( n > 0 \)) and \( u = b_1 \cdots b_m \) (\( m \geq 0 \)) say.
Step 2

Using registers

\[ Q = \text{current state} \]

\[ A = \text{current tape symbol} \]

\[ D = \text{current direction of tape head} \]

\[ \text{NB, } (\text{with } L = 0, \ R = 1 \text{ and } S = 2) \]

one can turn the finite table of (argument,result)-pairs specifying \( \delta \) into a RM program \( \rightarrow (Q, A, D) ::= \delta(Q, A) \rightarrow \) so that starting the program with \( Q = q, \ A = a, \ D = d \) (and all other registers zeroed), it halts with \( Q = q', \ A = a', \ D = d' \), where \( (q', a', d') = \delta(q, a) \).
Step 3

The next slide specifies a RM to carry out $M$’s computation. It uses registers

\[ C = \text{code of current configuration} \]

\[ W = \text{code of tape symbols at and left of tape head (reading right-to-left)} \]

\[ U = \text{code of tape symbols right of tape head (reading left-to-right)} \]

Starting with $C$ containing the code of an initial configuration (and all other registers zeroed), the RM program halts if and only if $M$ halts; and in that case $C$ holds the code of the final configuration.