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.
Turing machines, informally
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 ▶.

- 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 $\rhd$.
- 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 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 \( Q = \{s, q, q'\} \) (\( s \) 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>(rej,0,S)</td>
<td>(rej,1,S)</td>
</tr>
<tr>
<td>q</td>
<td>(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>(rej,(\triangleright),R)</td>
<td>(acc,(\sqcup),S)</td>
<td>(rej,0,S)</td>
<td>(q',1,L)</td>
</tr>
</tbody>
</table>
Turing machine configuration: \((q, w, u)\)

where

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

\((\text{So } wu \in \Sigma^* \text{ represents the current tape contents.})\)
Turing machine computation

Turing machine configuration: \((q, w, u)\)

where

- \(q \in Q \cup \{\text{acc, rej}\} = \) 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 $q \neq \text{acc, rej}$, $w = va$ (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
\end{array}
\begin{array}{c}
v \mid a \mid u
\end{array}
\xrightarrow{\sim} 

\begin{array}{c}
q'
\downarrow
\end{array}
\begin{array}{c}
v \mid a' \mid u
\end{array}

(q, va, u) \rightarrow_m (q', v, a'u)
\[ \delta(q, a) = (q', a', S) \]

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

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

\[
\begin{array}{c|c|c}
q & a & u \\
\hline

(\overrightarrow{va}, u) & \rightarrow^M & (q', ?, ?, ?)
\end{array}
\]
\[ \delta(q, a) = (q', a', R) \]

Two cases:

\[ \begin{cases} 
    u = a^* u^* & \text{is non-empty} \\
    u = \varepsilon & \text{is empty}
\end{cases} \]

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

\[
\begin{array}{c}
\begin{array}{c}
q \\
\downarrow \\
\begin{array}{c}
v \\
\mid \\
a \\
\mid \\
a''u''
\end{array}
\end{array}
\end{array}
\xrightarrow{\sim} 
\begin{array}{c}
\begin{array}{c}
q' \\
\downarrow \\
\begin{array}{c}
va' \\
\mid \\
a'' \\
\mid \\
u''
\end{array}
\end{array}
\end{array}
\]

\[
(q, va, a''u'') \rightarrow^* (q', va'a'', u')
\]

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

\[ (q, va, \varepsilon) \rightarrow^_m (q', va', \varepsilon) \]

Two cases:

\[ \begin{cases} u = a^\prime u^\prime \text{ is non-empty} \\ u = \varepsilon \text{ is empty} \end{cases} \]
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

states \( Q = \{s, q, q'\} \) (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 \((\text{acc}, \rhd \sqcup, 1^{n+1} 0)\).
Example Turing Machine

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

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

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

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

a string of \( n \) 1s

Claim: the computation of \( M \) starting from configuration \( (s, \triangleright, \sqsubseteq 1^n0) \) halts in configuration \( (\text{acc,} \triangleright, \sqsubseteq 1^{n+1}0) \).
The computation of $M$ starting from configuration $(s, \triangleright, \underline{1}^n0)$:

$$(s, \triangleright, \underline{1}^n0) \rightarrow_M (s, \triangleright_{\underline{1}}, 1^n0)$$

$$\rightarrow_M (q, \triangleright_{\underline{1}}1, 1^{n-1}0)$$

$$\vdots$$

$$\rightarrow_M (q, \triangleright_{\underline{1}}1^n, 0)$$

$$\rightarrow_M (q, \triangleright_{\underline{1}}1^n0, \varepsilon)$$

$$\rightarrow_M (q, \triangleright_{\underline{1}}1^{n+1}0, \varepsilon)$$

$$\rightarrow_M (q', \triangleright_{\underline{1}}1^{n+1}, 0)$$

$$\vdots$$

$$\rightarrow_M (q', \triangleright_{\underline{1}}, 1^{n+1}0)$$

$$\rightarrow_M (\text{acc}, \triangleright_{\underline{1}}, 1^{n+1}0)$$
The computation of $M$ starting from configuration $(s, \triangleright, \square 1^n0)$:

$$(s, \triangleright, \square 1^n0) \xrightarrow{M} (s, \triangleright \square, 1^n0)$$

$$\xrightarrow{M} (q, \triangleright \square 1, 1^{n-1}0)$$

$$\vdots$$

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

$$\xrightarrow{M} (q, \triangleright \square 1^n0, \varepsilon)$$

$$\xrightarrow{M} (q, \triangleright \square 1^{n+1} \square, \varepsilon)$$

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

$$\vdots$$

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

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

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

- 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.
reversal of \( w \) makes it easier to use our RM programs for list manipulation

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

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

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

\[ Q = \text{current state} \]
\[ A = \text{current tape symbol} \]
\[ D = \text{current direction of tape head} \]

(with \( L = 0 \), \( R = 1 \) and \( S = 2 \), say)

one can turn the finite table of (argument, result)-pairs specifying \( \delta \) into a RM program

\[ (Q, A, D) ::= \delta(Q, A) \rightarrow \]

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

\[
\begin{align*}
C &= \text{code of current configuration} \\
W &= \text{code of tape symbols at and left of tape head (reading right-to-left)} \\
U &= \text{code of tape symbols right of tape head (reading left-to-right)}
\end{align*}
\]

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.