Turing Machines

What do we mean by Computation???

The first third of the 20th century resulted in our, finally, obtaining a useful formal idea of what is meant by computation. We also obtained a number of results about what functions were "computable" and what kind of theorems were provable (and not) via an algorithmic approach.

Alan Turing was one of the pioneers, and provided us with an "abstract machine" whose computational power appears to correspond to the power of any algorithm - in fact we define computability in terms of such machines. Several other attempts were made at devising other definitions of computability (most notably by Alonzo Church, inventor of the Lambda-Calculus), up to the quantum methods that were begun in the 1980s. No abstract machine devised since Turing's is capable of computing any larger class of functions (quantum algorithms can compute faster, but that is their only - admittedly very important - advantage).

Turing Machines

Def.: A Deterministic Turing Machine is an abstract machine consisting of three parts: a tape, a read-write head operating on the tape, and a finite control (= a finite state machine that controls the behavior of the read-write head depending on the current state and the character currently under the read-write head).

Tape: divided into cells, is bounded "on the left" and unbounded "on the right". Each cell can hold a symbol from a finite tape alphabet \( \Gamma \).

\( \Gamma \): there are three kinds of symbols in \( \Gamma \): a) the input symbols \( \Sigma \), the blank symbol \( \square \) \( \notin \Sigma \), and the auxiliary symbols \( \Gamma - \{ \square \} - \Sigma \).

Tape Head: can move along the tape, one cell at a time, either to the right or to the left. It can read a symbol and erase a symbol and replace it by another one.

Finite Control: a finite state machine with two distinguished states - an initial state \( s \) and a final state \( h \). \( Q = \text{all states} \ - \ h \).

Turing Machines

Def.: a one-tape DTM is a quintuple \( M = (Q, \Sigma, \Gamma, \delta, s) \), where:

1. \( Q \) is a finite set of states.
2. \( \Sigma \) is a finite alphabet of input symbols.
3. \( \Gamma \) is a finite alphabet of tape symbols, \( \Sigma \cup \{ \square \} \subseteq \Gamma \).
4. \( s \) is the initial state.
5. \( \delta \) is a transition function

\[ \delta : Q \times \Gamma \rightarrow (Q \cup \{ h \}) \times \Gamma \times \{ L, R \} \]

which takes a state and a tape symbol and returns a new state, writes a new tape symbol in place of the old one and moves the read-write head one position to the left or right.
Turing Machines

How does a DTM start?
1. \( M \) is in the initial state \( x \).
2. The input \( x \) is stored in the \( 2^n \) to \( (n+1)^n \) cells of the tape, \( n = \text{tl} \).
3. All other cells contain the symbol B.

How do we describe the \textit{successive configurations}?
1. We need to know the current state \( q \).
2. We need to know the string on the tape.
3. We need to know the positions of the read-write head.

\[
(q, x_1 x_2 \ldots x_n, y, s) \quad \text{Tape before the read-write head}
\]
\[
\text{Character under the read-write head}
\]
\[
(q, x_1 x_2 \ldots x_n, y, s) \quad \text{Tape after the read-write head; blanks further to the right}
\]

Turing Machines

More terminology
1. At configuration \((q, x y)\), if \( b(q, a) = (q, b, R) \) and \( y \neq \varepsilon \), the move leads to the successor configuration \((p, x b_1 y)\) where \( y = y_1 y_2 \) and \( |y_2| = 1 \). We also denote this transition by \((q, x y) \xrightarrow{a} (p, x b_1 y)\).
2. At configuration \((q, x y)\), with \( b(q, a) = (p, h, R) \), the successor configuration is \((p, b y)\).
3. At configuration \((q, x y)\), if \( b(q, a) = (q, b, L) \) and \( x \neq \varepsilon \), the move leads to the successor configuration \((p, x a b y)\) if \( y \neq \varepsilon \) or \( b \neq B \), or it becomes \((p, x a b)\) if \( y = \varepsilon \) and \( b = B \), where \( z = x a b \) and \( |y_2| = 1 \).
4. At configuration \((q, y)\), with \( b(q, a) = (p, b, L) \), the machine hangs. No successor configuration.
5. At configuration \((b, x y)\), the machine halts and accepts the input. No successor configuration.

Turing Machines

4. If the head is to the right of the non-blank part of the string:
\[
(q, x_1 x_2 \ldots x_n B \ldots B, B, i)
\]

\text{Def.:} a \textit{configuration} of a DTM is a string in
\[
Q \times \Gamma^* \times \Gamma \times (\Gamma^* \setminus \{B\}) \cup \{\varepsilon\},
\]
which we denote by the quadruple \((q, x, a, y)\).

What this means for \( y \) is that it either is \( \varepsilon \) or ends in a non-blank. Another abbreviation for the configuration is \((q, x y)\), and a last is \( x q a y \).

The final abbreviation is unambiguous when \( Q \cap \Gamma = \emptyset \).

How do we describe the moves from configuration to configuration?

Turing Machines

Recall that a one-step transition was also denoted by \((q, x y) \xrightarrow{a} (p, y x b)\). An \( n \)-step transition \((n \geq 0)\) is denoted by \( a_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} \).

\text{Def.:} a string \( z \) is \textit{accepted} by a DTM \( M \) if \( M \) halts at state \( h \) on input \( x \). Equivalently, if \((x, B x y) \xrightarrow{a} (b, y z)\) for some \( y, z \in \Gamma^* \) and \( a \in \Gamma \).

\text{Def.:} the sequence of configurations \((x, B x y) \xrightarrow{a} \ldots \xrightarrow{a} (h, y z)\) is the \textit{computation path} of \( M \) on \( x \).

\text{Def.:} a string \( z \) is \textit{not accepted} by \( M \) if a) \( M \) hangs on \( x \) or \( M \) does not halt on \( x \).
Turing Machines

**Ex. 4.1.** $M_1 = (Q, \Sigma, \delta, s, a, b, \Gamma)$, where $Q = \{s, q_1, q_2, q_3, q_4, q_5\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, B\}$ and $\delta$ is given by the table below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$a$</th>
<th>$b$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>$p, a, R$</td>
<td>$p, b, R$</td>
<td>$q_5, B, L$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_2, a, L$</td>
<td>$q_2, b, L$</td>
<td>$q_2, B, R$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$p, a, R$</td>
<td>$q_5, b, R$</td>
<td>$h, B, L$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$p, a, R$</td>
<td>$p, b, R$</td>
<td>$p, B, L$</td>
</tr>
</tbody>
</table>

This machine will accept the language $L(M_1) = \{a^m b^n | m \geq 0, n \geq 1\} = a^*b^*b^*$. Try $(s, \text{BabbaB})$ and $(s, \text{BabbbB})$.

**Theorem 4.2.** Every regular language is Turing-acceptable.

**Proof.** Let $L$ be accepted by the DFA $M = (Q, \Sigma, \delta, s, F)$. From this construct a one-tape DTM $M' = (Q \cup \{s\}, \Sigma \cup \{B\}, N, s)$, where $s, t \notin Q$ and

- $\delta(s, B) = (t, B, L)$
- $\delta(t, a) = (t, a, L)$ if $a \notin \Sigma$
- $\delta(q, a) = (p, a, R)$ if $a \notin B$, $q \notin Q$ and $b(q, a) = p$
- $\delta(h, B) = (h, B, L)$ if $a = B$ and $q \in F$.

$M'$ first moves left until the leftmost symbol of the input, and then it simulates $M$ until it reads a blank, and then halts on $x$ in state $h \Rightarrow M$ halts on $x$ in a final state.

---

Turing Machines

**Def.:** for any DTM $M$, let $L(M) = \{x \in \Sigma^* | M$ accepts $x\}$. A language $L$ is called Turing-acceptable if $L = L(M)$ for some DTM $M$.

**Theorem 4.2.** Every regular language is Turing-acceptable.

**Proof.** Let $L$ be accepted by the DFA $M = (Q, \Sigma, \delta, s, F)$. From this construct a one-tape DTM $M' = (Q \cup \{s\}, \Sigma \cup \{B\}, N, s)$, where $s, t \notin Q$ and

- $\delta(s, B) = (t, B, L)$
- $\delta(t, a) = (t, a, L)$ if $a \notin \Sigma$
- $\delta(q, a) = (p, a, R)$ if $a \notin B$, $q \notin Q$ and $b(q, a) = p$
- $\delta(h, B) = (h, B, L)$ if $a = B$ and $q \in F$.

$M'$ first moves left until the leftmost symbol of the input, and then it simulates $M$ until it reads a blank, and then halts on $x$ in state $h \Rightarrow M$ halts on $x$ in a final state.

---

Turing Machines

**Def.:** a function $f: (\Sigma^*)^k \rightarrow \Sigma^*$ is a **partial function** if it can be undefined at some $(x_1, \ldots, x_k) \in (\Sigma^*)^k$; a function $f: (\Sigma^*)^k \rightarrow \Sigma^*$ is a **total function** if it is defined at every $(x_1, \ldots, x_k) \in (\Sigma^*)^k$.

**Def.:** $f$ is **defined at $(x_1, \ldots, x_k)$** is denoted by $f(x_1, \ldots, x_k)$ ↓, while $f$ is **undefined at $(x_1, \ldots, x_k)$** is denoted by $f(x_1, \ldots, x_k)$ ↑.
Turing Machines

**Def.:** a partial function \( f : (\Sigma^*)^k \rightarrow \Sigma^* \) is computed by a one-tape DTM \( M \) if

1. for every \( (s_1, \ldots, s_k) \in (\Sigma^*)^k \) such that \( f (s_1, \ldots, s_k) \) is defined,
   \((s, Bx_1Bx_2\ldots Bx_kB) \rightarrow_M^* (s', Bx_1Bx_2\ldots Bx_kB)\),
   where \( y = f (s_1, \ldots, s_k) \), and
2. for every \( (s_1, \ldots, s_k) \in (\Sigma^*)^k \) such that \( f (s_1, \ldots, s_k) \) is undefined,
   \((s, Bx_1Bx_2\ldots Bx_kB) \rightarrow_M^* \ldots \) never halts.

**Def.:** a partial function \( f : (\Sigma^*)^k \rightarrow \Sigma^* \) is called **Turing-computable** if it is computed by a one-tape DTM.

**Def.:** A language \( L \) is **Turing-decidable** if its characteristic function
\[
\chi_L (s) = \begin{cases} 
1 & \text{if } s \in L, \\
0 & \text{otherwise}, 
\end{cases}
\]
is Turing-computable (as a total function).

---

**Ex. 4.3.** \( A = \{ w w^R | w \in \{0, 1\}^* \} \) is Turing-acceptable.

**Soln.:** the basic idea is to compare \( s_1 \) to \( r_1, s_2 \) to \( r_2 \), etc. To do this we must set up states and transitions. Observe that the input alphabet has only 2 characters, for a total of 3 characters in the tape alphabet.

\[
\begin{array}{ccc}
\delta & 0 & 1 \\
\hline
s & q_1, B, L \\
q_1 & q_2, B, L \\
q_2 & q_3, 0, L \\
q_3 & q_4, 0, L \\
q_4 & q_5, B, R \\
q_5 & q_6, 0, R \\
q_6 & q_7, 1, R \\
q_7 & q_8, B, L \\
\end{array}
\]

---

**Ex. 4.4.** Show that the following function is Turing-computable:
\[ f(x) = w \text{ if } x = w w^R \text{ for some } w \in \{0, 1\}^* \]

\[ f(x) = \uparrow \text{ otherwise.} \]

**Soln.:** the problem is that we cannot quite erase all the matching characters - after all, we need to end up with a non-blank part of the tape containing \( w \). The idea is to replace the leading \( x \)'s with \( x \)'s. At the end of the match we simply change the remaining \( x \)'s back to \( x \)'s. Besides the states and transitions of the previous problem, we will need some extra states and extra transitions.

---

This corresponds to the machine (state-transition diagram):
We have the transition table:

<table>
<thead>
<tr>
<th>δ</th>
<th>0</th>
<th>1</th>
<th>0'</th>
<th>1'</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>q₁, B, L</td>
<td>q₂, B, L</td>
<td>q₃, 0, L</td>
<td>q₄, 1, L</td>
<td>h, B, R</td>
</tr>
<tr>
<td>q₁</td>
<td>q₂, 0, L</td>
<td>q₃, 1, L</td>
<td>q₄, 0, R</td>
<td>q₅, 1, R</td>
<td>q₆, B, R</td>
</tr>
<tr>
<td>q₂</td>
<td>q₃, 0, R</td>
<td>q₄, 1, R</td>
<td>q₅, 1, R</td>
<td>q₆, 0, R</td>
<td>q₇, B, L</td>
</tr>
<tr>
<td>q₃</td>
<td>q₄, 0, R</td>
<td>q₅, 1, R</td>
<td>q₆, 0, R</td>
<td>q₇, 1, R</td>
<td>q₈, B, R</td>
</tr>
<tr>
<td>q₄</td>
<td>q₅, 0, R</td>
<td>q₆, 1, R</td>
<td>q₇, 0, R</td>
<td>q₈, 0, R</td>
<td>q₉, B, L</td>
</tr>
</tbody>
</table>

Let us begin by modifying the original machine to use the $ symbol:

1. The input was empty to begin with - only Bs on the tape. In that case we go from the start state s to q₁, and immediately to q₉.
2. The input was not empty - and we succeed in replacing the right half of the string with Bs, and the left half with $s: on return from finding a B while going right, we immediately find a $: we have finished the matches and we now need to clean up.

Case 1 is simple: we just need to add an action that will replace the B under the read-write head (the second B on the tape) with a 1.
Case 2 is a little harder: we need to replace all the $s with Bs, until we find the leftmost B, then copy this leftmost B, move one position to the right, and replace the second B with a 1.

The modified machine appears on the next slide:
We still need to deal with the failure cases.

When $B$ is found in state $q_n$, it is time to blank out all cells to the left and replace the second blank from the left by a 0.

We now turn to several functions that may not seem well-motivated: we will need them later when we define the classes of primitive-recursive and recursive functions.

**Def.** let $\mathcal{N}$ denote the natural numbers, which we represent in unary format: $1^n = 111 \ldots 1$ $n$ times. We say that a partial function $f: \mathcal{N} \to \mathcal{N}$ is Turing-computable if the function $f : (\{1\}^*)^* \to (\{1\}^*)^*$ defined by $f(1^n, 1^{n_2}, \ldots, 1^{n_k}) = 1^{f(n_1, n_2, \ldots, n_k)}$ is Turing-computable.

The initial tape configuration will be: $B1^nB1^nB1^nB BBBB\ldots$ the final tape configuration will be: $B1^n1^n1^nB BBBB\ldots$
Ex. 4.6. The function \( n_1 n_2 = n_2 \), for \( n_1, n_2 \in \mathbb{N} \), is Turing-computable.

Proof: Recall that we must take the initial tape configuration \( B^1 B^2 \ldots \) and transform the tape to \( B^1 B^2 \ldots \). How? We set up a loop that will, at each iteration, reduce the first block of is by 1 and copy the second block of 1s one cell to the left. The TM that will do that is:

![Turing Machine Diagram]

Ex. 4.7. The function \( \text{sub}(n, m) = n - m \) if \( n \geq m \geq 0 \) and \( m > n > 0 \), is Turing-computable.

Solution: At each iteration, reduce both \( m \) and \( n \) by 1:

\[
(q_i, B^1 B^2 \ldots) \rightarrow (q_i, B^1 B^2 \ldots)
\]

The case \( n \geq m \geq 0 \) will be handled by the TM:

![Turing Machine Diagram]
If \( n < m \), at some point the machine will be in state \( q_1 \), with a B rather than a 1 under the read-write head. At that point, we must erase all the remaining 1s, and leave the read-write head on the second B from the left end of the tape:

3. We now need to erase the \( k - i \) blocks to the right of the "projection block". To perform this erasure, we need to reach the configuration \( (s_{out}, B_1, B_1, B_1, \ldots, B_{k-i+1}) \). This "counts down" on the erasures. To do so, we need a TM fragment:

![Turing Machine Diagram]

Note that the upper branch corresponds to "at least one more block to be erased", while the lower one corresponds to "no more blocks to be erased".

Ex. 4.8. For \( 1 \leq i \leq k \), show that \( \pi_i(x_1, x_2, \ldots, x_k) = x_i \) on \( \mathcal{N} \) is Turing-computable (this is the projection on the \( i \)-th component for \( k \)-dimensional vectors of natural numbers).

**Proof.** This is a little trickier than the textbook indicates - some details are "swept under the rug". We will try to re-introduce them (at least in part), and show ways to resolve them.

1. How do we represent things? A reasonable beginning would be to fill the left-hand part of the input tape with 
\[ y = B_1B_2B_3B_4B_5B_6B_7 \ldots, \]
so \((s, B_1B_2B_3B_4B_5B_6B_7B_8)\) is the initial configuration. The case \( \pi_i(n_1, n_2) \) did not need the extra input information because everything was fixed.

2. Using the \( \pi_i(n, m) \) function just shown to be Turing-computable we can compute 
\[(s, B_1B_2B_3B_4B_5B_6B_7B_8) \Rightarrow (h_{sub}, B_1B_2B_3B_4B_5B_6B_7B_8).\]

4. The removal of a block should take us along the computation path 
\((s_{out}, B_1B_2B_3B_4B_5B_6B_7B_8) \Rightarrow (h_{out}, B_1B_2B_3B_4B_5B_6B_7B_8).\) This can be done via use of the TM for the Turing-computable function \( \pi_i(n_1, n_2) \):
Turing Machines

5. So we are now at \((h_{\text{init}}, B_1 B_2 B_3 \ldots B_k i-1 B B)\). Repeat "count down followed by projection of second element" until we reach 
\((h_{\text{init}}, B_1 B_2 B_3 \ldots B_k BB)\). We now perform 
\((h_{\text{init}}, B_1 B_2 B_3 \ldots B_k BB) \rightarrow (h_{\text{init}}, B_1 B_2 B_3 \ldots B_k B)\).

At this point we can repeat the \(\pi_2\)-TM until we have removed all the 
blocks of 1s that precede \(x_i\). This brings up another problem: how do we 
avoid "falling off the left end"?

Note: the problem with the textbook approach is that it does not 
specify how we count the blocks we are dropping "on the right", and 
how we stop dropping blocks "on the left". Although the "handwaving" 
is intuitively compelling, the details are harder... And all because we 
are trying to use a single tape...

Ex. 4.9. The function
\[ \text{insert}(x_1, x_2, \ldots, x_k, y) = (x_1, \ldots, x_i \uparrow y, x_{i+1}, \ldots) \]
on strings over \(\{a, b\}\) is Turing-computable.

Proof. The textbook has a reasonable explanation. You should try to not 
only construct the two procedures but also determine what kind of 
"connective tissue" (in terms of TM fragments) you will need to make 
the ideas work.

Theorem 4.10. Every language accepted by a one-tape, read-only 
DTM is regular.

Proof. See the text. Since such a TM appears, to all intents and 
purposes, to be a DFA, and no more (does being able to back up give 
you anything extra?), this is intuitively plausible. The details of a 
formal proof are rather messy...

Turing Machines

Multi-Tape Machines

We now extend the idea of a DTM by extending the functionality of 
the tape. The first extension makes it impossible to "fall off the left end" 
of the tape, and ending up in an error condition.

Two-way "infinite" tape: the left-hand end of the tape is also 
unbounded (you can think of a tape, which although finite, can be 
extended as needed) and filled with \(B\)s.

What difference would this have made to our construction of a DTM 
implementing \(\pi(x_1, x_2, \ldots, x_k) = x_i\)? None, really: we still would have 
needed to keep track of the left end of the leftmost block, or to keep 
track of how many blocks we need to drop "on the left".

Two-Way-Tape Machines

Question: Is a DTM with a two-way unbounded tape more powerful 
than a standard DTM?

That it is at least as powerful should be obvious: if you use a 
standard DTM, your tape head may never move to the left of the first 
cell. So mark the cell containing the blank at the left of the input as 
"the leftmost one" (\(B\) should do it), and make sure that any transitions 
to the left of it result in a permanent new non-final state, while all 
other transitions treat it as a \(B\). Then every input accepted by the 
standard DTM will be accepted by the extended (marked) one and 
every input rejected (or not accepted) by the standard one will end up 
in a non-accepting state for the extended (marked) one. The same 
holds for functions computed by the TMs.

Is it strictly more powerful?
Turing Machines

Theorem 4.11. a) every function that is computed by a two-way unbounded one-tape DTM is Turing-computable (= computable by a DTM with a tape unbounded only on the right); 
b) every language that is accepted by a two-way unbounded one-tape DTM is Turing-acceptable (= acceptable by a DTM with a tape unbounded only on the right).

Proof:
1. The tape looks like ...BBBxBBB...
2. When the DTM M computes a function, it writes its output anywhere in the tape: \( (x_1, x_2, ..., x_n) \) is mapped to \( (y_1, y_2, ..., y_n) \) via \( (s, x_1Bx_2B...Bx_nB) \rightarrow M' (b, y_1By_2B...By_nB) \).
3. How do we simulate a two-way tape TM with a one-way tape one?

Turing Machines

Let us start with a TM \( M = (Q, \Sigma, \Gamma, \delta, q_0) \). We also have a TM \( M' \), with a one-way tape: \( M' = (Q', \Sigma, \Gamma', \delta', q_0') \) with the same input alphabet, but where \( \Gamma' = \Gamma \times \Gamma \cup \{ \$ \} \). Note that the top track is "reversed".

Let the cells on \( M' \) be denoted by \( C_1', C_2', C_3', ..., C_n' \), with the head over \( C_{n+1}' \). The configuration of the machines at start will be:

\[
\begin{align*}
M &: \quad B \quad B \quad y_1 \quad y_2 \quad y_3 \quad y_n \quad \ldots \quad y_1 \quad y_2 \quad B \\
M' &: \quad \$ \quad B \quad B \quad B \quad B \quad \ldots \quad B \quad B
\end{align*}
\]

We do not discuss the TM that we need to produce to change one setup into the other - it IS possible (mark beginning and end of input on \( M \) with a special character... etc.).

Turing Machines

Answer: add symbols. Consider symbol pairs \( (s, y) \in \Gamma \times \Gamma \). Put them on the tape as: \( x_1 \), \( x_2 \), with the other as the "bottom symbol". Fix a cell, symbol (say, \( \$ \)) in it. The one-way infinite tape will then look like:

\[
\begin{align*}
\$ & \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad B \quad B \\
& \quad y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5 \quad y_6 \quad \$ \\
& \quad \ldots
\end{align*}
\]

The bottom track simulates the "right-hand half" (cells \( C_1, C_2, ... \)), while the top track simulates the "left-hand half" (cells \( C_n, C_{n-1}, C_{n-2} \)).

Since the original states don't know about left-hand and right-hand, we must (at least) double the number of states: \( q \rightarrow q_0, q_1 \), one corresponding to "reading from the top", the other to "reading from the bottom".

Turing Machines

How do we change the states? As indicated: \( q \rightarrow q_2, q_3 \).

We start in \( s = x_1 \) (there is no \( y_1 \)). To construct \( \delta'(q_0, (B, B)) \) we make use of \( b(s, B) = \delta(q_0, (B, B)) = \delta(s, B) \). We remain in a state with subscript \( b \) (moving along the "regular states" of \( M \)) until our read-write head is over the symbol \( s \). That position was reached through a LEFT move, with state, say \( r_3 \). Since \( s \) did not exist in \( M \), we must add a transition \( \delta(r_3, S) = (r_4, S, R) \), and an "opposite transition" \( \delta(r_4, S) = (r_3, S, R) \).

What is the behavior in the \( r \)-subscripted states? Moving left in \( M' \) is equivalent to moving right in \( M' \), and moving right in \( M \) is equivalent to moving left in \( M' \).

At the (successful) end of the computation, \( M' \) will be in one of two states: \( h_a \) or \( h_b \). At that point we need to reconvert to \( M \) and this is possible if we know where the right-endpoint of the output is on \( M' \) - again, make sure that you have a special character there.
Turing Machines

Multi-Tape TMs.

The next question asks about the computational power of multi-tape TMs. As one would expect:

**Theorem:** for any \( k \geq 2 \), any TM with \( k \) two-way unbounded tapes is equivalent to a TM with a single two-way unbounded tape (and thus to a "standard" TM).

1. Each TM has two special tapes - an input tape and an output tape. The head of the input tape is read-only. The heads of the other tapes can read, erase and write. All heads are controlled (simultaneously) by a finite control.
2. During each move a head can move left, right or stay (= not move).
3. The transition function \( \delta : Q \times \Gamma^k \rightarrow (Q \cup \{L\}) \times \Gamma^k \times \{L, R\} \).
4. An instruction: \( \delta(q, (a_1, a_2, \ldots, a_k), x) = (p, (b_1, b_2, \ldots, b_k), (D_1, D_2, \ldots, D_k)) \).
5. A configuration: the state plus \( k \) tape configurations.

At the very beginning we expect an initial configuration:

\[ (s', [x_1, B, \ldots, B][x'_1, B, \ldots, B], \ldots, [x_n, B, \ldots, B]) \]

In particular, the single read-write head is to the right of all of the input and head-position symbols.

**How does the simulation work?**

1. Each state \( q \) of \( M \) becomes a collection of states \( q_{i,p_{12},q_{i,p_{2}}}, \) for all possible \( k \)-tuples of characters that can appear under the \( k \) read-write heads of \( M \). In fact, we need even more: add another character \( \$ \). So that each state becomes a set indexed by all \( k \)-length strings over \( \Gamma' \cup \{\$\}. The number of states is still finite.
2. Our first move to the left will take us to some state \( q_{i,p_{12},q_{i,p_{2}}}. As we keep moving to the left, we successively replace \$ characters from \( \Gamma' \) until we are in state \( q_{i,p_{12},q_{i,p_{2}}}. We are in state \( q_{i,p_{12},q_{i,p_{2}}}, \) and \( a_i \) is under the \( i^{th} \) read-write head for \( 1 \leq i \leq k \).

**Note:** in the case of the simulation of a two-way tape machine by a one-way one we still had to keep track of only one read-write head. In this case, even though one of the heads is read-only, we still need to keep track of \( k \) heads.

**Solution:** double the number of tracks on the single tape and use a special symbol to denote the head position on that tape:

\[
\begin{array}{ccccccc}
B & B & B & B & B & B & B \\
B & B & B & B & B & B & x \\
B & B & B & B & B & x & B \\
B & B & B & B & B & x & B \\
B & B & B & B & B & x & B \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
B & B & B & B & B & B & B \\
\end{array}
\]

Tape1 Head1 Tape2 Head2 Tape k Head k

3. Now we have the read-write head at the left of all individual tape head positions. At this point we can use the transition function \( \delta(q_{i,p_{12},q_{i,p_{2}}}, B) = \delta(q, (a_1, \ldots, a_k)) = (p, (b_1, \ldots, b_k), (D_1, \ldots, D_k)). \)
4. What are the actions of the single read-write head? We start moving to the right until we find the first (leftmost) symbol indicating the position of a multi-tape read-write head. We are in state \( q_{i,p_{12},q_{i,p_{2}}} \) and we now have \( (\ldots, a_j, x, \ldots) \) under the head. We must:
   1. Replace \$ with \( h \) on the \( j^{th} \) subtape.
   2. Erase the \( x \) on the second level of the \( j^{th} \) subtape.
   3. Move the head as \( D_j \) indicates.
   4. Write \( x \) on the second level of the \( j^{th} \) subtape.
   5. Move the head position back to where this started.
   6. If another head is positioned (lower) in the same column, repeat; otherwise move right.
There are several places where we must remember what we have done and ensure that the rest of the action completes correctly:

1. When subactions 1 and 2 are completed, state \( q_{1a_j} \) must change to state \( q_{2a_j} \) and then to state \( q_{3a_j} \) to indicate that no further actions on that subtape need to be carried out.
2. Note that we need a total of 4 state markers: \( a_j, a_j, L, a_j, R \) and \( \_a_j \).
5. Continue until we have changed each state marker \( a_j \) into the state marker \( \_a_j \). Move to the right until the end of the tape is found (all \( B \)). At that point the state \( q_{1a_1}, \ldots, q_{j_a_j}, \ldots, q_{k_a_k} \) is changed to \( p_\_ \), and we are ready for the next pass.
6. At the end, we must still ensure that the "output tape" contains the output.

The next reasonable question is: what if we have infinitely many (an unbounded number of) tapes? More precisely, we have:

\[ M = (Q, \Sigma, \Gamma, \delta, q_0) \]

where \( \delta : Q \times \Gamma^* \rightarrow (Q \cup \{h\}) \times \Gamma \times \{L, R, S\} \).

Note that \( \Gamma^* \) contains non-empty finite length strings, and we use them as though the characters were written one per tape (on the same "column", so to speak), rather than sequentially on a single tape. An instruction can then be denoted by

\[ \delta(q, a, b, \ldots, d, D_1, D_2, \ldots, D_n) \]

where all the missing characters are \( B \)s. Note that the number of characters changed may be more or less than the number of characters read, while the number of heads moved may be different from either other number. The only requirement is that all activities take place on contiguous tapes (= sequentially numbered from 1 at the top, going down).

### Example 4.12
Find a three-tape DTM \( M \) that computes the function \( f(n, m) = n \cdot m \).

**Solution:** Assume the initial configuration is \((\_1B^1B, B, B)\). \( M \) must construct a TM so that the final configuration of tape 3 is \( 1B^1B \).

**Algorithm:**
1. Copy \( 1^m \) to tape 2.
2. For each \( 1 \) on tape 2, delete it and copy \( 1^n \) from tape 1 to tape 3
3. When tape 2 has no \( 1 \)s, halt with output on tape 3.

### Example 4.14
For any language \( A \subseteq (0+1)^* \) there is an unbounded number of tapes DTM that accepts \( A \).

**Proof:**
1. Start with the string \( x = a_1a_2\ldots a_k \in (0+1)^* \) on the input tape - tape 1, say.
2. Copy, on tapes 2 through \( |x|+1 \), the individual characters from \( x \), in left-to-right order. At the beginning and end of this process, the tape should look like this:

\[ \_1B^1B \ldots \_1B^1B \Rightarrow^* \_1B^1B \ldots \_1B^1B \]

\[ \_1B^1B \ldots \_1B^1B \]

\[ \_1B^1B \ldots \_1B^1B \]
At this point, we move all the heads one position to the left, leaving the blanks in place and apply the transition function (one instruction).

The problem is that our transition function must correspond to infinitely many instructions, even though the number of states (and the length of each string) is still finite: we need finitely many states to perform the copy, and just one more state ($p$ besides $s$ and $h$) to finish.

We define the transition function:

$$\delta(p, [B, a_1, \ldots, a_k]) = (h, [B, B, \ldots, B], R)$$

if $a_1 \ldots a_k \in L$, and

$$\delta(p, [B, a_1, \ldots, a_k]) = (p, [B, B, \ldots, B], S)$$

if not.

NOTE: this does it, but, clearly, violates the requirement of finiteness for $\delta$. Even though it reflects the unbounded number of tapes format, there does not seem to be any way to avoid this jump from finite $\delta$ to infinite $\delta$.

Non-Deterministic TMs.

One might ask the question: what of Context-Free Languages? Regular languages are accepted by DFAs (which are DTMs), but CFLs are accepted by Non-Deterministic PDAs - one can show that Deterministic PDAs recognize a strictly smaller set of languages. If we could show that a Non-Deterministic TM can be simulated via a Deterministic one, we would also have that CFLs are accepted by DTMs.

Def.: A Non-Deterministic TM has a finite control and a single one-way infinite tape. For a given state and tape symbol scanned by the tape head, the machine has a finite number of choices for the next move. Each choice consists of a new state, a tape symbol to print, and a direction of head motion. The Non-Deterministic TM accepts its input if any sequence of choices of moves leads to an accepting state.

Theorem. If $L$ is accepted by a NDTM $M_1$, then $L$ is accepted by some DTM $M_2$.

Proof. The crucial point is that $\delta(p, a)$ has bounded cardinality for any choice of $p \in Q$ and $a \in \Gamma$. Let this least upper bound be $r$, where each available move from each pair $(p, a)$ is numbered by $0 \leq j \leq r$ (some move sets may have strictly fewer than $r$ moves).

$M_2$ has 3 tapes: 1) the input tape (tape 1); 2) a tape (tape 2) on which $M_2$ will generate finite sequences of numbers in the range $1 \ldots r$, systematically from length 1 on; 3) a tape (tape 3) on which, for each sequence, $M_2$ will simulate $M_1$ on the input along the sequence of operations encoded on tape 2.

It should be clear that $M_1$ accepts $\iff$ $M_2$ accepts.

Corollary: CFLs are Turing acceptable.

Church-Turing Thesis.

What do we mean by Computation? We could define it in terms of one-tape TMs and be done, BUT we would be begging the question of whether there is another model of computation that is, in some sense, more powerful.

The first thing we can do is establish what we mean by a reasonable model of computation:

1. Computation of the machine is given by a finite set of instructions.
2. Each instruction can be carried out in a finite number of steps, or in a finite amount of time.
3. Each instruction is deterministic (the transition function is a function).
The Church-Turing thesis states that any reasonable computational model computes exactly the same set of functions that are computable by TMs.

This is not a theorem, nor even a formal conjecture, since it is based on an intuitive comprehension of what is meant by "reasonable model of computation". At this point, at least, all "reasonable models of computation" that have been formalized have been proven equivalent to TMs.

We will look at one such model (based on Noam Chomsky's "generative grammars") in the next section; the Lambda Calculus (that gave rise to LISP and to the family of functional languages) is another; Quantum Computation is a third.