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

Deterministic Turing Machine

Def.: A **Deterministic Turing Machine** is an abstract machine consisting of three parts:

- **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 ($\mathbb{B} \notin \Sigma$), and the auxiliary symbols $\Gamma - \{\mathbb{B}\} - \Sigma$.
- **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

Deterministic Turing Machine

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, $\Sigma \cup \{B\} \subseteq \Gamma$.
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.

![Diagram of a one-tape DTM](image-url)
Turing Machines

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

How do we describe the 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.

$$(q, x_1 x_2 \ldots x_{k-1}, x_k, x_{k+1} x_{k+2} \ldots x_m)$$

- state
- Tape before the read-write head
- Character under the read-write head
- Tape after the read-write head; blanks further to the right
4. If the head is to the right of the non-blank part of the string:

\((q, x_1x_2 \ldots x_mB\ldots B, B, \varepsilon)\)

**Def.**: a *configuration* of a DTM is a string in

\[Q \times \Gamma^* \times \Gamma \times (\Gamma^*(\Gamma - \{B\}) \cup \{\varepsilon\}),\]

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 \((q, xay)\), and a last is \(xqay\). The final abbreviation is unambiguous when \(Q \cap \Gamma = \emptyset\).

How do we describe the moves from configuration to configuration?
Turing Machines

More terminology: Configurations

1. At configuration \((q, xay)\), if \(\delta(q, a) = (p, b, R)\) and \(y \neq \varepsilon\), the move leads to the *successor configuration* \((p, xb_{y_1}y_2)\) where \(y = y_1y_2\) and \(|y_1| = 1\). We also denote this transition by \((q, xay) \rightarrow_M (p, xb_{y_1}y_2)\).

2. At configuration \((q, xa)\), with \(\delta(q, a) = (p, b, R)\), the successor configuration is \((p, xb\mathbb{B})\).

3. At configuration \((q, xay)\), if \(\delta(q, a) = (p, b, L)\) and \(x \neq \varepsilon\), the move leads to the *successor configuration* \((p, x_1x_2by)\) if \(y \neq \varepsilon\) or \(b \neq \mathbb{B}\), or it becomes \((p, x_1x_2)\) if \(y = \varepsilon\) and \(b = \mathbb{B}\), where \(x = x_1x_2\) and \(|x_2| = 1\).

4. At configuration \((q, ay)\), with \(\delta(q, a) = (p, b, L)\), the machine hangs. No successor configuration.

5. At configuration \((h, xay)\), the machine halts and accepts the input. No successor configuration.
More terminology: Acceptance

Recall that a one-step transition was also denoted by $(q, xay) \vdash_M (p, xby_1y_2)$. An $n$-step transition ($n \geq 0$) is denoted by $\alpha_1 \vdash_M^* \alpha_n$.

**Def.** a string $x$ is **accepted** by a DTM $M$ if $M$ halts at state $h$ on input $x$. Equivalently, if $(s, BxB) \vdash_M^* (h, yaz)$ for some $y, z \in \Gamma^*$ and $a \in \Gamma$.

**Def.** the sequence of configurations $(s, BxB) \vdash_M^* (h, yaz)$ is the **computation path** of $M$ on $x$.

**Def.** a string $x$ is **not accepted** by $M$ if a) $M$ hangs on $x$ or $M$ does not halt on $x$. 
Turing Machines

Deterministic Turing Machine: Example

Ex. 4.1. \( M_1 = (Q, \Sigma, \Gamma, \delta, s) \), where \( Q = \{s, q_1, q_2, q_3, p\} \), \( \Sigma = \{a, b\} \), \( \Gamma = \{a, b, B\} \) 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_1, B, L)</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_1, a, L)</td>
<td>( q_1, b, L)</td>
<td>( q_2, B, R)</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_2, a, R)</td>
<td>( q_3, b, R)</td>
<td>( p, B, L)</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( p, a, R)</td>
<td>( q_3, b, R)</td>
<td>( h, B, L)</td>
</tr>
<tr>
<td>( p )</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 \((s, B a b b a B)\) and \((s, B a b b b b B)\).
Deterministic Turing Machine: Example

As with DFAs and PDAs, DTMs can be represented via transition diagrams: the states (a finite set) are represented by (named) circles and the transitions by labeled arrows. The machine of Ex. 4.1 has a transition diagram:

![Transition Diagram](image-url)
Def.: for any DTM $M$, let $L(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$. A language $L$ is called **Turing-acceptable** if $L = L(M)$ for some DTM $M$.

What languages are Turing-acceptable?

1. Regular Languages should, since a DFA looks like a simplified Turing Machine: it should be possible to formalize the relationship.

2. Context-Free Languages should, since a Pushdown Automaton should be implementable on a Turing Machine - we are pushing TMs as the ultimate model of computation. The only problem is non-determinism (a non-deterministic PDA is strictly more powerful than a deterministic one: see Hopcroft & Ullman, *Introduction to Automata Theory, Languages and Computation* (1979), p. 113 & Ex. 5.7, p. 121).

3. There should be some more classes of languages that are accepted by TMs... e.g., the Chomsky Hierarchy, otherwise grammars would be more powerful that TMs.
Turing Machines

Turing-acceptable: the Easy Case

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

**Proof.** Let $L$ be accepted by the DFA $M = (Q, \Sigma, \delta, q_0, F)$. From this construct a one-tape DTM $M' = (Q \cup \{s, t\}, \Sigma, \Sigma \cup \{B\}, \delta', s)$, where $s, t \notin Q$ and

\[
\begin{align*}
\delta'(s, B) &= (t, B, L) \\
\delta'(t, a) &= (t, a, L) \text{ if } a \in \Sigma \\
&= (q_0, B, R) \text{ if } a = B \\
\delta'(q, a) &= (p, a, R) \text{ if } a \neq B, q \in Q \text{ and } \delta(q, a) = p \\
&= (h, B, L) \text{ if } a = B \text{ and } q \in F.
\end{align*}
\]

$M'$ 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 $h \iff M$ halts on $x$ in a final state.
Turing Machines

**Turing acceptable; Turing decidable; Turing computable**

**Def.** the non-blank symbols (if any) left on the tape when $M$ halts in state $h$ are called the **output** of $M$.

Recall that a (partial) function is a set of pairs $(x, y)$ where $x$ is the "input to the function" and $y$ is the "output of the function". We can thus associate the idea of function with the idea of Turing Machine: the first element of the pair is the input string, while the string left on the tape at successful conclusion is the output one.

**Def.** a function $f : (\Sigma_1^*)^k \rightarrow \Sigma_2^*$ is a **partial function** if it can be undefined at some $(x_1, \ldots, x_k) \in (\Sigma_1^*)^k$; a function $f : (\Sigma_1^*)^k \rightarrow \Sigma_2^*$ is a **total function** if it is defined at every $(x_1, \ldots, x_k) \in (\Sigma_1^*)^k$.

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

**Def.** a partial function $f : (\Sigma_1^*)^k \rightarrow \Sigma_2^*$ is computed by a one-tape DTM $M$ if

1. for every $(x_1, \ldots, x_k) \in (\Sigma_1^*)^k$ such that $f(x_1, \ldots, x_k)$ is defined,
   $$(s, Bx_1Bx_2B\ldots Bx_kB) \vdash_M^* (h, ByB),$$
   where $y = f(x_1, \ldots, x_k)$, and

2. for every $(x_1, \ldots, x_k) \in (\Sigma_1^*)^k$ such that $f(x_1, \ldots, x_k)$ is undefined,
   $$(s, Bx_1Bx_2B\ldots Bx_kB) \vdash_M^* \ldots \text{ never halts.}$$

**Def.** A language $L$ is **Turing-decidable** if its characteristic function

$$\chi_L(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{otherwise,} \end{cases}$$

is Turing-computable (**as a total function**).
Turing Machines

Turing-acceptable: Example

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

Soln.: the basic idea is to compare $x_n$ to $x_1$, $x_{n-1}$ to $x_2$, 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.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>0</th>
<th>1</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td></td>
<td></td>
<td>$q_1, B, L$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_2, B, L$</td>
<td>$q_4, B, L$</td>
<td>$h, B, R$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_2, 0, L$</td>
<td>$q_2, 1, L$</td>
<td>$q_3, B, R$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_6, B, R$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_4$</td>
<td>$q_4, 0, L$</td>
<td>$q_4, 1, L$</td>
<td>$q_5, B, R$</td>
</tr>
<tr>
<td>$q_5$</td>
<td></td>
<td></td>
<td>$q_6, B, R$</td>
</tr>
<tr>
<td>$q_6$</td>
<td>$q_6, 0, R$</td>
<td>$q_6, 1, R$</td>
<td>$q_1, B, L$</td>
</tr>
</tbody>
</table>
Turing Machines

This corresponds to the machine (state-transition diagram):
Turing Machines

Turing-acceptable: Example - DPDA vs NPDA

Recall that \( A = \{ w w^R \mid w \in \{0, 1\}^* \} \) is a CFL.
We can show that in two ways:
1. By constructing a CFG that generates the language;
2. By constructing a Non-Deterministic PDA that accepts it.

The grammar construction does not tell us much. The NPDA construction points out that we have no way to decide WHEN we have reached the middle of the string, so we can start unwinding the stack: that is why we need non-determinism. Although this is not quite a proof that no D(deterministic)PDA can be found, it should be enough for all practical purposes.

**Questions**: can we construct a 2DPDA (a 2-stack DPDA) that accepts it? More generally, what languages are equivalent to 1DPDAs?
Turing Machines

Turing Computable: Example

Ex. 4.4. Show that the following function is Turing-computable:

\[ f(x) = \begin{cases} w & \text{if } x = w w^R \text{ for some } w \in \{0, 1\}^* \\ \uparrow & \text{otherwise.} \end{cases} \]

Solln.: 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 \( x_i \)'s with \( x'_i \)'s. At the end of the match we simply change the remaining \( x'_i \)'s back to \( x_i \)'s.

The input alphabet is \( \{0, 1\} \); the tape alphabet is \( \{0, 1, B, 0', 1'\} \). Besides the states and transitions of the previous problem, we will need some extra states and extra transitions.
### Turing Machines

#### Turing Computable: Example

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

Turing Decidable: Example

Ex. 4.6. \( \{w w^R \mid w \in \{0, 1\}^* \} \) is Turing-decidable.

Soln.: we need to leave a 0 (no) or a 1 (yes) on the second cell of the tape. We can't just use the TM of Ex. 4.3, since that replaces everything with blanks, and then we would not know which is the "second from the left" blank to be replaced by a 1 (having accepted). Furthermore, failure results in being in states from which we cannot exit (\( q_3 \) and \( q_5 \) on having other than a 0 or a 1 - respectively - under the read-write head). Those conditions must then lead us to erasing the whole tape, leaving a 0 in the second cell. To know when the beginning of the tape is found, we will use a \$\ (and a \( B \) for those from the right).
Turing Machines

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

What are the characteristics of success?
1. The input was empty to begin with - only Bs on the tape. In that case we go from the start state $s$ to $q_1$ and immediately to $q_h$.

2. The input was not empty - and we succeed in replacing the right half of the string with Bs, and the left half with $s$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 $s$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:
Turing Machines

Success:

We still need to deal with the failure cases.
Turing Machines

1. We can fail by encountering a 1 or B in state $q_3$.
2. We can fail by encountering a 0 or B in state $q_5$.
3. All other states have successful exits for all possible inputs.
Turing Machines

When B is found in state $q_8$, it is time to blank out all cells to the left and replace the second blank from the left by a 0.
Turing Machines

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

The initial tape configuration will be: $B1^{n_1}B1^{n_2}B\ldots B1^{n_k}BBB\ldots$
the final tape configuration will be: $B1^{f(n_1, n_2, \ldots, n_k)}BBB\ldots$
**Ex. 4.6.** The function $\pi_2^2(n_1, n_2) = n_2$, for $n_1, n_2 \in \mathbb{N}$, is Turing-computable.

*Pf.* Recall that we must take the initial tape configuration $B1^{n_1}B1^{n_2}B\ldots$ and transform the tape to $B1^{n_2}B\ldots$ How? We set up a loop that will, at each iteration, reduce the first block of 1s by 1 and copy the second block of 1s one cell to the left. The TM that will do that is:

![Turing Machine Diagram]
Turing Machines

We can see how this works on the example:

\[(s, B11B111B) \to (q_1, B11B111B) \to (q_2, B11B11BB) \to^* (q_2, B11B11BB) \to (q_3, B1111BB) \to (s, B1B111BB) \to^* (s, B1B111BB) \to (q_1, B1B111B) \to (q_2, B1B11BB) \to^* (q_2, B1B11BB) \to (q_3, B111BB) \to (s, BB111B) \to^* (s, BB111B) \to (q_2, BB11BB) \to (q_3, B111BB) \to (q_4, B111BB) \to^* (q_4, B111BB) \to (q_5, B111BB) \to (h, B111BB)\]
Ex. 4.7. The function

\[ sub(n, m) = \begin{cases} 
  n - m & \text{if } n \geq m \geq 0 \\
  0 & \text{if } m > n \geq 0,
\end{cases} \]

Is Turing-computable.

*Soln.*: at each iteration, reduce both \( m \) and \( n \) by 1:

\[(q_1, B1^i B1^{i-1} \underline{1}) \vdash^* (q_1, B1^{i-1} B1^{i-2} \underline{1}).\]

The case \( n \geq m \geq 0 \) will be handled by the TM:
We can run the example \texttt{sub}(3, 2):

\[
(s, B111B11B) \vdash (q_1, B111B11B) \vdash (q_2, B111B1BB) \vdash^* (q_2, B111B1BB) \vdash (q_3, B111B1BB) \vdash (q_4, B11B1BB) \vdash^* (q_4, B11B1BB) \vdash (q_5, B11B1BB) \vdash (q_1, B11B1BB) \vdash (q_2, B11BBB) \vdash (q_3, B11BBB) \vdash (q_4, B1B1BB) \vdash (q_4, B1B1BB) \vdash (q_5, B1B1BB) \vdash (q_1, B1BB) \vdash (q_8, B1BB) \vdash (h, B1BB)
\]
Turing Machines

If $n < m$, at some point the machine will be in state $q_3$ 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:
**Turing Machines**

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

\textit{Pf.} 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

   $$y = Bx_1Bx_2B\ldots Bx_kBkBiBBB\ldots,$$

   so $(s, Bx_1Bx_2B\ldots Bx_kBkBiB)$ is the initial configuration. The case $\pi_2^2(n_1, n_2)$ did not need the extra input information because everything was fixed.

2. Using the $\text{sub}(n, m)$ function just shown to be Turing-computable we can compute

   $$(s, Bx_1Bx_2B\ldots Bx_kBkBiB) \xrightarrow{*_{\text{sub}}} (h_{\text{sub}}, Bx_1Bx_2B\ldots Bx_kBk-iB).$$
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 $(s_{block}, Bx_1Bx_2B\ldots Bx_kBk-i-1B)$. This "counts down" on the erasures. To do so, we need a TM fragment:

![Diagram of Turing Machine]

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".
4. The removal of a block should take us along the computation path \((s_{\text{block}}, Bx_1Bx_2B...Bx_kBk-i-1B) \vdash^* (h_{\text{sub}}, Bx_1Bx_2B...Bx_{k-1}Bk-i-1B)\). This can be done via use of the TM for the Turing-computable function \(\pi_2^2(n_1, n_2)\):
Turing Machines

5. So we are now at \((h_{sub}, Bx_1Bx_2B\ldots Bx_{k-i}Bk-i-1B)\). Repeat "count down followed by projection of second element" until we reach
\((h_{sub}, Bx_1Bx_2B\ldots Bx_iBB)\).

We now perform \((h_{sub}, Bx_1Bx_2B\ldots Bx_iBB) \vdash (h_i, Bx_1Bx_2B\ldots Bx_iB)\). 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"? We can solve this problem by adding the $ character at the very beginning of the tape. We are done dropping blocks when we read the $. At that point we can simply move the remaining block \(B11\ldots 1B = Bx_iB\) one place to the left and we are done.

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

\[ \text{insert}_i^k(x_1, x_2, \ldots, x_k, y) = (x_1, \ldots, x_{i-1}, y, x_i, \ldots, x_k), \quad 1 \leq i \leq k+1, \]
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…