1 Turing machines

A Turing Machine (TM) is a relatively simple mathematical model, which still captures the full power of general purpose computing.

**Definition 1.1 (Turing machine).** A turing machine has an input tape, an output tape and at least one work tape. Tapes are divided into cells, each of which can record a single symbol. The input tape is read-only, the remaining are both read and write. There is a reading head on each tape which can be moved one step in each operation. It also has a finite state which can be used to represent code, registers, etc.

A k-tape Turing Machine is given by the tuple $M = (\Gamma, Q, \delta)$ where

- $\Gamma$ is the alphabet (a finite set of symbols, e.g. ASCII).
- $Q$ is a finite set of internal states (e.g. code, location in code, registers).
- $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ represents the state change, tape rewrite and tape head moves given current state and characters under the reading heads.

**Definition 1.2 (Configuration).** A configuration of a TM is given by its current state, the contents of the tapes and the location of the reading heads on each tape.

We assume for simplicity that for any TM, $Q$ always has two special states: $q_{\text{start}}$ and $q_{\text{end}}$ and that the alphabet $\Gamma$ always contains a “blank” symbol, as well as 0, 1 which are the input alphabet.

**Definition 1.3 (Computing a function by a Turing machine).** Turing machines compute partial functions $f : \{0,1\}^* \rightarrow \{0,1\}^* \cup \{\text{undefined}\}$, where

- The TM is initialized in state $q_{\text{start}}$, input tape with an input $x \in \{0,1\}^*$ padded by blanks, and the remaining tapes blank.
- The TM is run.
• If the TM reaches the state $q_{\text{end}}$ the content of the output tape is taken to be $f(x)$, otherwise $f(x) = \text{undefined}$.

**Definition 1.4** (Running time). A TM $M$ has running time $T: \mathbb{N} \to \mathbb{N}$ if

- $M(x)$ always halts for all $x \in \{0,1\}^*$.
- $M$ makes at most $T(|x|)$ steps when running on input $x$.

The main thing we will show is that the model doesn’t matter. That is, with only small increases in running time, we can simulate any reasonable variant of a Turing Machine with a “standardized TM” that has exactly 3 tapes and a binary alphabet.

**Lemma 1.5** (The alphabet doesn’t matter). If $f: \{0,1\}^* \to \{0,1\}^* \cup \{\text{undefined}\}$ can be computed by a TM with alphabet $\Gamma$ it can also be computed with a TM with alphabet $\{0,1,\text{blank}\}$.

*Proof.* Assume $\Gamma = 2^c$. Let $M$ be a TM computing $f$ with alphabet $\Gamma$. We will simulate $M$ by a TM $M'$ which has alphabet $\Gamma' = \{0,1\}$ and a few more states. Formally, assume $M$ has $W_1, \ldots, W_k$ as contents of the tapes, reading heads at locations $i_1, \ldots, i_k$ and state $q \in Q$. Then

- Each symbol in the tapes of $M$ is encoded by $c$ bits.
- The reading heads in $M'$ locations are in the first bit of the corresponding symbols, e.g. $c \cdot i_1, \ldots, c \cdot i_k$.
- $M'$ reads the $c$ bits in each location and stores in internal registers (part of state space $Q'$ of $M'$).
- $M'$ simulates the change of state of $M$, writes the corresponding bits encoding the new symbols, and moves the heads accordingly.

At the end, the configuration of $M'$ encodes the next configuration of $M$. Note that if $M$ runs in time $T(n)$ on inputs $x \in \{0,1\}^n$ then $M'$ runs in time $O(T(n) \log |\Gamma|)$. $\square$

**Lemma 1.6** (The number of tapes doesn’t matter). If $f: \{0,1\}^* \to \{0,1\}^* \cup \{\text{undefined}\}$ can be computed by a TM with $k \geq 3$ tapes. Then $f$ can also be computed with a TM with three tapes (input, work and output).

*Proof.* Let $M$ be the $k$-tape TM computing $f$. We simulate $M$ using three tapes as follows:

- Encode reading head location in each tape as part of the input (increase alphabet size by a constant).
- Interleave working tapes of $M$ on a single work tape of $M'$. 

2
• To simulate a step in $M$: initially all reading heads of $M'$ are at the beginning. First, $M'$ scans for the reading heads of all tapes and saves the content (as part of its internal state). Then, it simulates the change in $M$, by scanning the tape again and making the appropriate changes.

If $x \in \{0, 1\}^n$ is the input of $M$ and runs in time $T(n)$, then the tapes have at most $kT(n)$ non blank symbols, hence each step if $M$ is simulated by $O(kT(n))$ steps in $M'$. Hence $M'$ runs in time $O(kT(n)^2)$.

A machine with a RAM can read a word from memory by a single instruction. These too can be simulated by Turing machines.

**Lemma 1.7** (RAM doesn’t matter). If $f : \{0, 1\}^* \to \{0, 1\}^* \cup \{\text{undefined}\}$ can be computed by a TM with RAM then it can also be computed by a standard TM.

**Proof.** Simulate the RAM read by scanning the working tapes. Increases time from $T$ to $O(T^2)$.

The simulations we saw increase runtime by at most squaring it. So the models discussed above are all equivalent up to a quadratic factor in the runtime.

## 2 Universal machines

We can encode a TM $M$ by a string $\langle M \rangle \in \{0, 1\}^*$ representing its “program” (e.g. state space, alphabet and transition function). A universal machine is simply an interpreter for these programs. We say a Turing machine $U$ taking two inputs is a universal machine if,

$$ U(\langle M \rangle, x) = M(x), $$

and if $M$ loops forever so does $U$. if $\langle M \rangle$ is not a legal program (does not compile) output some default output.

**Lemma 2.1.** *There exist universal Turing machines.*

**Proof.** Assume for simplicity$^1$ $M$ has a single work tape and has $\Gamma = \{0, 1, \text{blank}\}$. $U$ will have two work tapes: one that is a copy of the worktape of $M$, and another auxiliary work tape for internal computation of the “interpreter” $U$. The second work tape will be used to remember the state of $M$. To simulate a step in $M$:

- Read input from work tape of $M$.

- Scan the transition function (given in the input) to match the state of $M$ (given in the auxiliary work tape). Write the new state of $M$ in the auxiliary work tape, and implement the changes in the work tape of $M$ that we maintain (change bit value, move head).

---

$^1$By the above discussion showing that alphabet and number of tapes does not matter, this assumption is without loss of generality.
If $M(x)$ runs in time $T(n)$ then $U(\langle M \rangle, x)$ runs in time $O(|\langle M \rangle| \cdot T(n))$.

It can be shown (though we won’t show it here) that even if $M$ has more than one work tape, it can still be simulated by a single work tape universal machine in time $O(T(n) \log T(n))$.

### 3 Uncomputable functions

Not all functions are computable. Define a function $HALT : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ as

$$HALT(\langle M \rangle, x) = \begin{cases} 0 & \text{If } M(x) \text{ halts} \\ 1 & \text{If } M(x) \text{ doesn’t halt, or } \langle M \rangle \text{ is not a valid code} \end{cases}$$

**Theorem 3.1.** $HALT$ is not computable by Turing machines.

**Proof.** Assume $HALT$ is computable by a Turing machine. Then so is the following partial function

$$g(x) = \begin{cases} 0 & \text{If } h(x) = 1 \\ \text{undefined} & \text{If } h(x) = 0 \end{cases}$$

Assume $g$ is computable by a Turing machine $M$ and then reason by cases on the value of $g(\langle M \rangle)$.

- If $g(\langle M \rangle) = 0$ then $h(\langle M \rangle, \langle M \rangle) = 1$, meaning $M(\langle M \rangle)$ never halts, so $g(\langle M \rangle)$ should be undefined.
- If $g(\langle M \rangle) = \text{undefined}$ then $h(\langle M \rangle, \langle M \rangle) = 0$, meaning $M(\langle M \rangle)$ halts, so it must be that $g(\langle M \rangle) = 0$.

We see that $g(\langle M \rangle)$ cannot be defined, meaning that $g$ (and $h$) cannot have a TM computing them.

There are many uncomputable functions. Let’s show another one — the busy beaver function. For this one it would be convenient to consider the following variant of a TM without input. These machines have alphabet $\Gamma = \{0,1\}$ and just one tape (R/W). The initial configuration is where the tape is all zeros, the head is in the beginning of the tape, and the machine is in the start state. When the machine halts, the output is the content of the tape (removing any suffix of all zeros). Define the Busy Beaver function $BB : \mathbb{N} \rightarrow \mathbb{N}$ as the maximal number of 1’s which can be written by a halting TM with at most $n$ states.

**Theorem 3.2.** The busy beaver function is uncomputable.

**Proof.** Assume $BB$ can be computed by a TM $M$ which has $k$ states. Let $M_{\text{DBL}}$ be a TM which, running on a tape with content $1^n0^*$, doubles the number of ones, e.g. ends in $1^{2n}0^*$. Assume $M_{\text{DBL}}$ has $\ell$ states. Let $n \geq 1$ be a parameter to be chosen later, and consider the following TM $N$: 

---

4
1. Write $n$ ones on the tape and move the head back to the beginning. Tape = $1^n 0^*$. 

2. Apply $M_{DBL}$. Now, tape = $1^{2n} 0^*$. 

3. Apply $M$. Now, tape = $1^{BB(2n)} 0^*$. 

4. Add one more 1. Now, tape = $1^{BB(2n) + 1} 0^*$. 

The TM $N$ outputs $BB(2n) + 1$ ones. Let’s see how many states are needed to implement it. Step 1 requires $n + O(1)$ states (the $n$ states to write the $n$ ones, the additional $O(1)$ states to move the head back to the beginning at the end). Step 2 requires $\ell$ states. Step 3 requires $k$ states. Step 4 requires additional $O(1)$ states. So: we build a machine with $n + k + \ell + c$ states, where $c = O(1)$, such that it outs $BB(2n) + 1$ many ones. Let $r = k + \ell + c$, and recall that we can still choose $n$ freely. We have

$$BB(n + r) \geq BB(2n) + 1 \quad \forall n \geq 1.$$ 

But if we choose $n \geq r$ we reach a contradiction, as $BB$ is a monotone function. 

Many other mathematical problems can be shown to be incomputable. Examples:

- Rice’s theorem: The only computable functions $f : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}$ for which $f(\langle M \rangle, x)$ depends just on $\langle M \rangle(x)$ are the constant functions.

- Generalized Collatz conjecture (3n+1) conjecture.

- Hilbert’s 10 problem is such: given a polynomial with integer coefficients, find if it has an integer solution.