1 Definition of a Turing machine

We fix an alphabet $A = \{B = \sigma_0, \sigma_1, \ldots, \sigma_k\}$ where $B$ denotes blank. We also fix a symbol $*$ which will be used to indicate the left edge of a tape.

A Turing Machine over the alphabet $A$ consists of the following:

1. A tape, which consists of infinitely many cells, each of which contains a symbol from $A \cup \{\ast\}$.
   The tape has a left edge, and is infinite to the right, with the cells indexed by the natural numbers (although the machine itself does not have access to these indices). The left-most cell (i.e. cell number 0) always contains the symbol $\ast$, and no other cell contains this symbol. At any given time, only finitely many cells will contain a symbol other than $B$. Equivalently, we can think of the tape as holding a string from $A^\ast$.

2. A read/write head, which moves along the tape, able to read the contents of the cell it is scanning and to alter the contents of this cell.

3. A finite set of states, $Q = \{q_0, q_1, \ldots, q_t\}$. At any instant the machine is in a single state. The machine starts in the initial state, $q_0$, and halts when it reaches the halt state, $q_t$.

4. A transition table which controls the action of the machine. This is a function $I : Q \times (A \cup \{\ast\}) \rightarrow Q \times (A \cup \{\ast\}) \times \{L, 0, R\}$, which when given the state of the machine and the contents of the cell scanned by the head, tells what state to change to, what symbol to write on the current cell, and which direction to move. The machine is not allowed to move left off the left edge of the tape; it is not allowed to change the $\ast$ at the left edge; and it is not allowed to write $\ast$ anywhere other than the leftmost cell.

The machine operates as follows. It is loaded with some initial tape configuration which is a function from $\mathbb{N}$ to $A \cup \{\ast\}$ with the above conventions. It starts in state $q_0$ with the head at the leftmost cell of the tape and proceeds step-by-step according to the transition table, halting if and only if it enters the halt state.

Definition 1 We say that a Turing machine $M = \langle A, Q, I \rangle$ computes an $n$-ary partial function $f : ((A \setminus \{B\})^n) \rightarrow (A \setminus \{B\})^*$ if, when started with a tape consisting of $\ast w_1 B w_2 B \ldots B w_n B B B \ldots$, the machine halts with the head at the left edge and the tape containing $\ast f(w_1, \ldots, w_n) B B B \ldots$ whenever $f(w_1, \ldots, w_n)$ is defined, and does not halt when started $s$ above in case $f(w_1, \ldots, w_n)$ is not defined.

A partial function is Turing-computable or TM-computable if there is a Turing machine which computes it.

We will eventually show the following:

Theorem 2 A partial function is TM-computable if and only if it is partial recursive.
For now, we show:

**Proposition 3** Every While-computable function is TM-computable.

The proof is by induction on program construction. For convenience, we use multi-tape Turing machines, using one tape to hold the contents of each register used by a While Program. Assignment statements are easily implemented; composition is achieved by concatenating machines (inserting the initial state of the next machine into the halt state of the previous); and While tests are achieved by checking whether a tape contains all blanks (easily done).

**Corollary 4** Every partial recursive function is TM-computable.

### 2 Coding Turing machines

Let $M = \langle A, Q, I \rangle$ be a Turing machine. We intend to code it and its operation so as to show that TM-computable functions are partial recursive. Below, $(\ldots)$ refers to the primitive recursive coding function for finite sequences introduced earlier.

- A **tape description** is a function $\tau : \mathbb{N} \setminus \{0\} \to A$ which is equal to $B$ for all but finitely many values. We identify $\tau$ with the function $\tau : \mathbb{N} \setminus \{0\} \to \{0, \ldots, k\}$ where we replace $\sigma_i$ by $i$. Its code is:

  $$\langle \tau \rangle = p_1^{\tau(1)} \cdot p_2^{\tau(2)} \cdots p_m^{\tau(m)}$$

  where $\tau$ is equal to $B$ for all $n \geq m$. Note that this is well-defined, i.e. we get the same value for larger choices of $m$, since $B = \sigma_0$ is coded by $p^0 = 1$ in the product.

- A **situation description** is $(q_i, \tau, k)$ where $q_i \in Q$, $\tau$ is a tape description, and $k \in \mathbb{N}$ is the position of the head on the tape. Its code is $\langle i, \langle \tau \rangle, k \rangle$.

- If $t_{i,j} = (q_i, \sigma_j, q_k, \sigma_l, \tilde{m})$ is an instruction in $I$ (with $q_i, q_k \in Q$, $s_j, s_k \in A \cup \{\ast\}$, $\tilde{m} \in \{L, 0, R\}$), then its code is

  $$\langle t_{i,j} \rangle = \langle i, j, k, l, m \rangle$$

  where $m = 0, 1, 2$ as $\tilde{m} = L, 0, R$, respectively, and $\ast$ is considered to be $\sigma_{k+1}$.

- The **code of a Turing machine** $M$ is

  $$\langle M \rangle = \langle k, p, \langle t_{0,0} \rangle, \ldots, \langle t_{0,k+1} \rangle, \ldots, \langle t_{p,0} \rangle, \ldots, \langle t_{p,k+1} \rangle \rangle$$

**Proposition 5** The following are all primitive recursive:

1. $TM(m) \iff m$ is the code of a Turing machine
2. $\text{TapeDesc}(m, t) \iff TM(m) \land t = \langle \tau \rangle$ for a tape description $\tau$ in the alphabet of $m$
3. $\text{Sit}(m, s) \iff TM(m) \land s$ is a situation description for $m$
4. Step\((m, s)\) = the situation description after running \(m\) for one step, starting in configuration \(s\) if Sit\((m, s)\), and 0 otherwise.

5. Initial\((m, s)\) ⇐⇒ Sit\((m, s)\) ∧ \(s\) codes an initial configuration.

6. Comp\((m, c)\) ⇐⇒ TM\((m)\) ∧ \(c\) is a finite sequence of situation descriptions for \(m\), with \((c)_0\) an initial configuration.

7. Final\((m, c)\) ⇐⇒ Comp\((m, c)\) ∧ \(c\) codes a halting computation, i.e. the final situation description is in the halt state.

8. \(T_n(m, c, x_1, \ldots, x_n)\) ⇐⇒ Final\((m, c)\) ∧ \(c\) is a computation starting from \(*1^{x_1}B \cdots B1^{x_n}\), i.e. \(c\) codes a halting computation for the machine \(m\) computing a function with inputs \(x_1, \ldots, x_n\).

**Theorem 6** Every TM-computable function is partial recursive.

**Proof:** Let \(f(x_1, \ldots, x_n)\) be computed by the Turing machine \(M\) with code \(m\). Let \(U\) be the primitive recursive function which, when given \(c\) coding a computation, tells the word on the tape of the final situation description (which will be the value computed by the TM). Then:

\[
f(x_1, \ldots, x_n) = U(\mu c [T_n(m, c, x_1, \ldots, x_n)])
\]

This gives:

**Theorem 7 (Kleene’s Normal Form Theorem)** There is a primitive recursive function \(U\) and primitive recursive relations \(T_n\) for \(n \geq 1\) such that for every \(n\)-ary partial recursive function \(f\) there is a number \(e\) such that

\[
f(x_1, \ldots, x_n) = U(\mu y [T_n(e, y, x_1, \ldots, x_n)])
\]