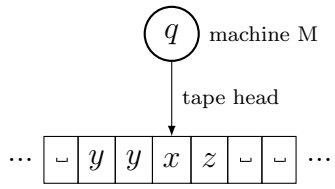

Figure 1.1: A Turing machine. The internal state is q, and the machine is currently reading the square marked x.

1 Turing Machines

Informally speaking, a Turing machine is a computer which possesses a finite number of internal states, and a one dimensional 'tape' as memory. We adopt the convention that the tape is unbounded in both directions. The tape is divided into individual squares each of which contains some symbol from a fixed alphabet; at any instant only one square is being read by the 'tape head'. Depending on the symbol on this square and the current internal state, the machine will write a symbol to the square under the tape head, possibly change the internal state and possibly move the tape head one square left or right. Formally,

Definition 1.1. A **Turing machine** $M = (\Sigma, Q, \delta)$ is a tuple where Q is a finite set of states, Σ is a finite set of symbols called the **tape alphabet**, and

$$\delta : \Sigma \times Q \to \Sigma \times Q \times \{\text{left, right, stay}\}$$

is a function, called the **transition function**.

The machine M is assumed to contain some designated symbol $\text{␣} \in \Sigma$ called the **blank symbol**, which is the only symbol allowed to occur infinitely often on the tape. One also assumes that there is a **starting state** $q_{\text{start}} \in Q$, along with a **halting state** $q_{\text{halt}} \in Q$.

Definition 1.2. Let $M = (\Sigma, Q, \delta)$ be a Turing machine, and let $w \in \Sigma^*$. We say M **halts** on input w if, when simulated on a tape which initially contains only the word w, the machine eventually reaches the state q_{halt}. If M halts on w, we write $M(w)$ to mean the non-blank contents of the tape upon halting; otherwise we leave $M(w)$ undefined. In this way, M can be considered as a partial function $M : \Sigma^* \to \Sigma^*$. One says that a partial function $F : \Sigma^* \to \Sigma^*$ is **computable** if there exists a Turing machine whose associated partial function is F.

One of the surprising features of the Turing machine model is that adding additional features does not increase its computing power. It may speed up execution, but does not change the set of functions which are computable. For example, consider a variant of the Turing machine model wherein a machine may have more than one tape, with

the tape heads moving independently. Formally, this means that the transition function becomes

$$\delta : \Sigma^n \times Q \to \Sigma^n \times Q \times \{\text{left, right, stay}\}^n$$

for some $n \geq 1$.

Lemma 1.3. Any multi-tape Turing machine can be simulated by a single-tape Turing machine.

Proof. Let $M = (\Sigma, Q, \delta)$ be an n-tape Turing machine. The idea we will use is to design a single-tape machine which print all tapes on top of one another, in the sense that the kth square of the single-tape machine will contain each of the kth squares of the n-tape machine. The main complication is that we also need to simulate of each of the head positions. The action of the single-tape machine then consists of a loop with two parts: we traverse the tape from left to right to find each symbol which is under a tape head, and then traverse the tape from right to left, updating the symbols, and shifting each of the simulated tape heads the appropriate direction.

The single-tape machine has tape alphabet

$$(\Sigma \times \{\sqcup, h\})^n,$$

with the blank symbol $((\sqcup, \sqcup), \ldots, (\sqcup, \sqcup))$. The set of states is

$$\underbrace{((\Sigma \cup \{?\})^n \times Q)}_{\text{Used in scan phase}} \cup \underbrace{((\Sigma \cup \{?\})^n \times Q \times \{\text{left}, \text{right}, \text{stay}, \text{update}\}^n)}_{\text{Used in update phase}}$$

where ? is assumed to not be a symbol in Σ. Given this, the behaviour of the single-tape machine is as follows. We begin in state $(?, ?, \ldots, ?, q_{\text{start}})$.

- **Scan phase**. The machine traverses the tape once from left to right. Any time a symbol such as $((\sigma_1, \sqcup), \ldots, (\sigma_i, h), \ldots, (\sigma_n, \sqcup))$ is observed, the ith '?' in the state is changed to the symbol σ_i. In this way, the machine keeps track of each of the symbols under the simulated tape heads. Once the machine has reached a state of the form $\mathbf{q} = (\sigma_1, \ldots, \sigma_n, q)$ where each $\sigma_i \in \Sigma$, the machine enters the update phase.

- **Update phase**. The update phase begins by transitioning to the state

  $$\delta(\mathbf{q}) =: (\sigma_1', ..., \sigma_n', q', d_1, ..., d_n).$$

  Our task is now to traverse the tape from right to left, updating the symbols and head positions. To do this, every time we encounter a symbol such as $((\sigma_1, \sqcup), \ldots, (\sigma_i, h), \ldots, (\sigma_n, \sqcup))$, the machine does the following:

  – If $d_i =$ 'stay', we simply update the pair (σ_i, h) to (σ_i', h), and then continue moving left.

– If $d_i =$ 'left' (resp. 'right'), we update the pair (σ_i, h) to $(\sigma_i', \sqcup)$, change d_i to 'update', and then move the tape head left (resp. right)[1].
– Finally, if at any point one of the d_i is 'update', we print an h at the corresponding position, and change d_i to 'stay'.

During this process, whenever a given symbol σ_i is updated on the tape to σ_i', we change the corresponding symbol in the state back to '?'. Once all of the symbols have been updated, we change our state to $(?, ?, \ldots, ?, q')$ and return to the scan phase.

□

2 Universal Turing Machines

A Turing machine U is universal if it can simulate any other Turing machine M when supplied with a tape containing a description of M. By changing the contents of the description tape, a universal Turing machine can therefore be used to compute *any* computable function. The remainder of the talk will be devoted to outlining one possible construction. The input to U will simply contain a list of tuples which defines the transition function of M, along with a section of the tape on which the state of M and tape contents are written. The operation of U then involves repeatedly locating the appropriate tuple and updating the state and tape accordingly.

One obstacle is that one must encode the simulated states using a series of symbols (for example, using binary) on the tape.

Definition 2.1. Let $M = (\Sigma_M, Q_M, \delta_M)$ and $U = (\Sigma_U, Q_U, \delta_U)$ be Turing machines. A **translation** of M for U is an injective function $T : \Sigma_M \cup Q_M \to \Sigma_U^*$ such that for all $x, y \in \Sigma_M \cup Q_M$ we have $|T(x)| = |T(y)|$.

In other words, a translation is a way of encoding each symbol and each state of M as some fixed length word in the language Σ_U^*. Note that a translation T can be lifted to a function $T : \Sigma_M^* \cup Q_M \to \Sigma_U^*$; given $w = w_1 w_2 \ldots w_k \in \Sigma_M^*$, we define

$$T(w) = \$\, T(w_1)\, \$\, T(w_2)\, \$ \ \ldots \ \$\, T(w_k)\, \$$$

where $\$ \in \Sigma_U$ is some designated punctuation symbol which is not used by T.

Definition 2.2. A Turing machine $U = (\Sigma_U, Q_U, \delta_U)$ is **universal** if, given any Turing machine $M = (\Sigma_M, Q_M, \delta_M)$, there exists a translation $T : \Sigma_M \cup Q_M \to \Sigma_U^*$ and a word $c_M \in \Sigma_U^*$ such that for any word $w \in \Sigma_M^*$ we have $U(c_M \sqcup T(w)) = T \circ M(w)$. We call c_M the **code** for M.

[1] If we have to update multiple heads at once because they are at the same position, this poses no problem. The only case we need to consider is when one head needs to move left, and the other right. In this case, we simply update the head which moves right first and ignore the head which moves left. Once the head which moves right has been updated, we continue scanning the tape from right to left, encounter the head which had to move left again, and update it then.

Theorem 2.3. There exists a universal Turing machine U.

Proof. We will describe a three-tape Turing machine U, whose tapes will be called the **description** tape, the **state** tape, and the **working** tape respectively. The tape alphabet of U is

$$\Sigma_U = \{\llcorner, 0, 1, \#, \$\},$$

and the set of states of U is

$$Q_U = \{\, \text{halt, compSymbol, compState, nextTuple, updateSymbol,} \\ \text{updateState, updateDir, resetDescr} \,\}.$$

We will defer describing the transition function δ_U for the moment.

Given a Turing machine M, define the translation $T : \Sigma_M \cup Q_M \to \Sigma_U^*$ by sending each element of $\Sigma_M \cup Q_M$ to some unique binary string, each of the same length k. The code c_M for M is defined as follows: for each $(\sigma, q) \in \Sigma_M \times Q_M$, let $(\sigma', q', d) = \delta_M(\sigma, q)$, and define $S_{\sigma,q} \in \Sigma_U^*$ as the word

$$T(\sigma)\,\$\,T(q)\,\$\,T(\sigma')^{\text{rev}}\,\$\,T(q')^{\text{rev}}\,\$\,d\,\#.$$

where a^{rev} means the reversal of a, and where the direction d is encoded as a sequence of $k+1$ zeroes to mean 'left', $k+1$ ones to mean 'right', and the empty string to mean 'stay'. The code c_M is then the concatenation of all such words $S_{\sigma,q}$.

Suppose that we wish to simulate M on input $w \in \Sigma_M^*$. Let q be the starting state of M. The machine U should begin with the word c_M on its description tape, $T(q)$ on the state tape, and $T(w)$ on the working tape. Each tape head should begin on the leftmost non-blank symbol, and the starting state of U is 'compSymbol'. Given this, the operation of U is as follows:

- The state 'compSymbol' compares the word encoding the current symbol on the working tape with the first tuple on the description tape, by reading both from left to right. This is performed by checking each bit in turn, until we either find a bit in which they differ (in which case we enter the state 'nextTuple'), or we reach a \$ on the description tape (in which case we go to 'compState').

- The state 'compState' is similar; we compare the word on the state tape with the tuple on the description tape. If they differ we go to 'nextTuple', otherwise 'updateSymbol'.

- The state 'nextTuple' moves the tape head on the description tape right until the symbol # is encountered, as well as moving the other two tape heads left until a blank is encountered. This has the effect of resetting the machine to prepare for checking the next tuple on the description tape. Following this, all three tape heads move right and we return to 'updateSymbol'.

- The state 'updateSymbol' then copies the third word of the tuple onto the working tape. Note that when the state 'updateSymbol' is reached, the tape head on the description tape is on the immediate left of the symbol to copy, while the tape head on the working tape is on the immediate right of the current simulated symbol. It is for this reason that the word encoding the symbol on the description tape is reversed; in order to copy from the description tape to the working tape, the description tape head is moving right while the working tape head is moving left. The state 'updateState' is similar. We than transition to 'updateDir'. At this point the working tape head is over the \$ immediately to the left of the current simulated symbol.

- The state 'updateDir' reads the word on the description tape which corresponds to the direction to move. For each symbol 0 we move the working tape head left, and for each symbol 1 we move the working tape head right. Once the description tape head reads the symbol #, we enter the state 'resetDescr'.

- The state 'resetDescr' simply moves the tape head on the description tape left until a blank symbol is encountered. At this point, we simply move all tape heads right, and return to the state 'compSymbol'.

□

3 References

1. M. Minsky, *Computation: finite and infinite machines*, Prentice-Hall Inc., Englewood Cliffs N.J., 1967.

2. M. Sipser, *Introduction to the theory of computation*, Thomson Course Technology, Boston, 2006.

3. A. Turing, *On Computable Numbers, with an Application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, **s2-42**, 1937, 230–265.