Discrete Math

Turing Machines

P. Danziger

1 Turing Machines

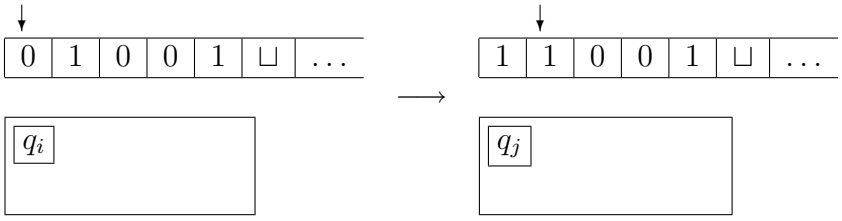
A Turing machine consists of a Finite State Control, which is an FSA, and an infinitely long read write 'tape'. This tape is divided into cells, at each step the read/write head is positioned over a particular cell.

The tape alphabet of a Turing Machine has a special symbol, often denoted $\sqcup$, or $\flat$, which indicates that a cell on the tape is blank.

A Turing Machine has two special states q_{accept} and q_{reject}.
If the machine ever enters the "accept" state, q_{accept}, it signals acceptance and halts processing.
If the machine ever enters the "reject" state, q_{reject}, it signals reject and halts processing.
Note that the only way that a Turing machine will halt is by entering one of these states, so it is possible that a Turing machine will continue processing forever and never halt.

Initially the tape contains the input string, and the tape read/write head is positioned over the leftmost symbol of the input string. At each step the Turing Machine performs the following actions:

1. Reads the current symbol from the tape.
2. Writes a symbol to the tape at the current position.
3. Moves to a new state in the Finite State Control.
4. Moves the read/write head either left or right one cell.



Note Unlike FSA's there is no requirement that a Turing machine read the input string sequentially, even if it does it may continue computing indefinitely (until it enters either q_{accept} or q_{reject}).

Definition (Deterministic Turing Machine) A Turing Machine, M, is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are finite sets and:

1. Q is the set of states of M.
2. Σ is the input alphabet of M. The blank symbol $\sqcup \notin \Sigma$.
3. Γ is the tape alphabet of M. $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$.
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function of M.

5. $q_0 \in Q$ is the start state of M.
6. $q_{\text{accept}} \in Q$ is the accept state of M.
7. $q_{\text{reject}} \in Q$ is the reject state of M.

Initially the tape contains the input string, and the tape read/write head is positioned over the leftmost symbol of the input string. The rest of the tape is filled with the blank symbol ($\sqcup$). The first blank thus marks the end of the initial input string.

The transition function then tells the machine how to proceed:

$$\delta(q_i, a) = (q_j, b, L)$$

Means: If we are in state q_i and we read an $a \in \Gamma$ at the current tape position, then move the finite state control to state q_j, write $b \in \Gamma$ and move the tape head left. (R means move the tape head right.)

Examples

1. $M_1 = (\{q_0, q_{\text{accept}}, q_{\text{reject}}\}, \{0,1\}, \{0,1,\sqcup\}, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$

   $\delta(q_0, 0) = (q_{\text{accept}}, 0, R)$
   $\delta(q_0, 1) = (q_{\text{reject}}, 1, R)$
   $\delta(q_0, \sqcup) = (q_{\text{reject}}, 1, R)$.

   This machine goes to the accept state if 0 is the first character of the input string, otherwise it goes to reject.

2. $M_2 = (\{q_0, q_1, q_{\text{accept}}, q_{\text{reject}}\}, \{0,1\}, \{0,1,\#,\sqcup\}, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$

   $\delta(q_0, 0) = (q_{\text{accept}}, 0, R)$
   $\delta(q_0, 1) = (q_1, 1, R)$
   $\delta(q_0, \#) = (q_{\text{reject}}, 1, R)$
   $\delta(q_0, \sqcup) = (q_{\text{reject}}, 1, R)$.

   $\delta(q_1, 0) = (q_1, \#, R)$
   $\delta(q_1, 1) = (q_1, \#, R)$
   $\delta(q_1, \#) = (q_1, \#, R)$
   $\delta(q_1, \sqcup) = (q_1, \#, R)$.

   (This can be summarized as $\forall a \in \Gamma, \delta(q_1, a) = (q_1, \#, R)$.)

   This machine goes to the accept state if 0 is the first character of the input string, it goes to the reject state if the input string is empty. If the string starts with a 1, it tries to fill up the infinite tape with #'s, which takes forever.

We can represent the action of a Turing machine on a given input by writing out the current tape contents, with the state to the left of the current read/write head position.

Thus if we consider the action of M_2 on the string 10001:

$$\begin{array}{llllll} q_0 10001\sqcup & \longrightarrow & 1q_1 0001\sqcup & \longrightarrow & 11q_1 001\sqcup & \longrightarrow \\ 111q_1 01\sqcup & \longrightarrow & 1111q_1 1\sqcup & \longrightarrow & 11111q_1 1\sqcup & \longrightarrow \\ 111111q_1\sqcup & \longrightarrow & 1111111q_1\sqcup & \ldots & & \end{array}$$

Definition

1. A string $w \in \Sigma^*$ is accepted by a Turing machine M if the machine enters the q_{accept} state while processing w.

2. The language $L(M)$ accepted by a Turing machine M is the set of all strings accepted by M.

3. A string is rejected by a Turing machine M if either M enters the q_{reject} state while processing w, or if M never halts in the processing of w.

4. A language L is called Turing recognizable or Recursively Enumerable if there exists some Turing machine, M, such that $L = L(M)$.

5. A language L is called Turing decidable or Recursive if there exists some Turing machine, M, such that $L = L(M)$, and M is guaranteed to halt on any input.

Note The difference between Recognizable and Decidable is that in the latter case the machine is guaranteed to halt.

The problem of showing that there are languages which are Recognizable but not Decidable, is essentially the halting problem.

Examples

1. Consider the machines M_1 and M_2 above.

   Clearly $L(M_1) = L(M_2) = 0(0 \vee 1)^* = L =$ any string beginning with 0.

   M_1 decides L, since it is guaranteed to stop.
   On the other hand M_2 only recognizes L, since it does not halt on any string beginning with a 1.

2. Design a Turing machine to decide the language $L = 0^n 1^n$.

   We introduce a tape character '#', to denote that the original symbol in a cell has been used. By saying it is 'crossed off' we mean replaced by '#'.

   $Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{accept}}, q_{\text{reject}}\}, \Sigma = \{0, 1\}, \Gamma = \{0, 1, \#, \sqcup\}$

   Algorithm:
   (a) Cross off the leftmost 0.
   (b) Scan right to end of input, cross off the rightmost 1.
   If there is no such 1 reject.
   (c) Scan left until we reach the leftmost surviving 0.
   If there is no such 0 scan right, if a 1 is encountered before reaching the end of the string reject, otherwise accept.

   $\delta(q_0, \sqcup) = (q_{\text{accept}}, \sqcup, R)$ – Accept the empty string
   $\delta(q_0, 1) = (q_{\text{reject}}, 1, R)$ – Reject any string which starts with 1
   $\delta(q_0, 0) = (q_1, \#, R)$ – Cross off leftmost 0

   $$\left.\begin{array}{rcl} \delta(q_1, 0) & = & (q_1, 0, R) \\ \delta(q_1, 1) & = & (q_1, 1, R) \\ \delta(q_1, \#) & = & (q_2, \#, L) \\ \delta(q_1, \sqcup) & = & (q_2, \sqcup, L) \end{array}\right\} \text{Scan right to rightmost 'live' cell of input.}$$

$\delta(q_2, \#) = (q_{\text{reject}}, \#, L)$ – If rightmost not 1 reject
$\delta(q_2, 0) = (q_{\text{reject}}, 0, L)$ – If rightmost not 1 reject
$\delta(q_2, 1) = (q_3, \#, L)$ – Cross off rightmost 1.

$\delta(q_3, 1) = (q_3, 1, L)$ – Scan left over the 1's
$\delta(q_3, 0) = (q_3, 0, L)$ – Scan left over the 0's
$\delta(q_3, \#) = (q_4, \#, R)$ – # marks left end

$\delta(q_4, 0) = (q_1, \#, R)$ – cross off leftmost 0 and start again.
$\delta(q_4, \#) = (q_{\text{accept}}, \#, R)$ – all done accept.
$\delta(q_4, 1) = (q_{\text{reject}}, 1, R)$ – No more 0's, but still got 1's.

The following should never be encountered, they are all set to go to q_{reject}.

$\delta(q_0, \#), \delta(q_2, \sqcup), \delta(q_3, \sqcup), \delta(q_4, \sqcup)$.

We now consider the action of this machine on the string 0011.

$$
\begin{array}{lllll}
q_0 0011\sqcup & \longrightarrow & \#q_1 011\sqcup & \longrightarrow & \#0q_1 11\sqcup \\
\longrightarrow & \#01q_1 1\sqcup & \longrightarrow & \#011q_1\sqcup & \longrightarrow \\
\#01q_2 1\sqcup & \longrightarrow & \#0q_3 1\#\sqcup & \longrightarrow & \#q_3 01\#\sqcup \\
\longrightarrow & q_3\#01\#\sqcup & \longrightarrow & \#q_4 01\#\sqcup & \longrightarrow \\
\#\#q_1 1\#\sqcup & \longrightarrow & \#\#1q_1\#\sqcup & \longrightarrow & \#\#q_2 1\#\sqcup \\
\longrightarrow & \#q_3\#\#\#\sqcup & \longrightarrow & \#\#q_4\#\#\sqcup & \longrightarrow \text{ Accept}
\end{array}
$$

We can see from the above that it is tedious to write out the transition function in full. Turing machines are very powerful, in fact the Church Turing thesis holds that they can implement any algorithm.

Often it is sufficient to write out an algorithm which describes how the Turing machine will operate. Of course if we are asked for a formal description, we **must** provide the transition function explicitly.

When writing out algorithms a common phrase is:
Scan right (or left) performing action until x is reached.

3. Design a Turing machine which decides $L = \{x \in \{0\}^* \mid x = 0^{2^n}, n \in \mathbf{N}\}$. (See Sipser p. 131 for a formal description.)

   $\Sigma = \{0\}, \Gamma = \{0, \#, \sqcup\}$.

   Scan right along the tape crossing off every other 0, this halves the number of 0's.
   If there is only one 0, accept.
   If the number of 0's is odd reject. (Note the parity of 0's can be recorded by state.)

   Scan back to the left hand end of the string.

   Repeat.

   Consider the action of this algorithm on the strings 00000000 (0^8), and 0000000 (0^7):

initially	00000000	initially	0000000
After first pass	# 0 # 0 # 0 # 0	After first pass	# 0 # 0 # 0 #
After second pass	# # # 0 # # # 0	After second pass	# # # 0 # # #
After third pass	# # # # # # # 0	After third pass	# # # # # # #
Accept		Reject	

The Halting Problem Given a Turing machine M does M halt on every input w?

Variations

There are several standard variations of the definition of Turing Machines which we will now investigate.

Multi-tape Turing Machines

A Multi-tape Turing Machine is a Turing machine which has more than one tape.

If the machine has k tapes, it is called a k tape Turing Machine.

The only change is in the transition function, we read k inputs from the k tapes, and write k outputs, in addition each of the k read/write heads moves either Left or Right.

$$\delta \, : \, Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R\}^k$$

Theorem If a language is recognized by some k tape Turing machine M, then it is recognized by some 1 tape Turing machine M'.

S.W.P.

Computation of Integer Functions

In this variation $\Sigma = \{0\}$, the initial input is of the form 0^n, for some n. When the machine halts the tape holds the computed *output*, 0^m, for some m. Such a machine is called a Computational Turing machine.

Thus for each n there is a corresponding m, we may interpret this as the result of some function: $m = f(n)$.

We say that the function f is computable.

A further variation allows for more than one input variable. In this case $\Sigma = \{0, \#\}$ and input strings are of the form $0^n \# 0^k$. The output, 0^m is then the result of $m = f(n, k)$

In this case the numbers n, k and m are being represented in unary notation, which is standard. However, it is possible to use $\Sigma = \{0, 1, \#\}$, and to represent the numbers n, k and m in binary.

Example Find a computational Turing machine which computes $f(n, m) = n + m + 1$

The input is of the form $0^n 1 0^m$, we merely erase the 1 and check that the input has the correct form.

$\delta(q_0, 0) = (q_0, 0, R)$ – Scan right for 1
$\delta(q_0, 1) = (q_1, 0, R)$ – Found it, change it to a 0
$\delta(q_1, 0) = (q_1, 0, R)$ – Scan right for end of string
$\delta(q_1, \sqcup) = (q_{\text{accept}}, \sqcup, R)$ – Found it, accept
Every other transition goes to q_{reject}.

It is worth noting that for all the many variations of Turing machines that have been investigated, **none** are more powerful (i.e. able to recognize more languages) than a standard Turing machine.

The Church Turing Thesis

Turing machines are extremely powerful in their computational abilities. They can recognize addition, subtraction, multiplication and division, exponentiation, (integer) logarithms and much more.

Any operation which a PC can do can be done on a Turing machine, In fact a Turing machine is far more powerful than any real computer since it effectively has an infinite amount of memory.

The Church-Turing Thesis says essentially that:

Any real computation (algorithm) can be simulated by a Turing machine.

It should be noted that the Church Turing thesis is an axiom, it is the link between the mathematical world and the real world. No amount of mathematics can ever prove this thesis because it states a fact about the real world.

Thus Turing machines represent the ultimate model of computation, if a language is not recognizable by a Turing machine, NO algorithm can compute it.

There are problems which are known to have now algorithmic solution. i.e. problems which are *unrecognizable*.

The Halting Problem

We first consider algorithms which are not decidable.

There are many important problems which are known to be undecidable.

For example the problem of software verification (verifying that a program works as specified) is undecidable.

When we talk about Turing machines in a general sense, as we do now, it is unproductive to worry about the internal workings of the machine (specifying state tables etc.). We are more interested in a general description.

In general Turing machines may be given a general object for analysis. All that is required that the general object be rendered in a form that the Turing machine can interpret, a finite string of characters from a suitable alphabet.

We may now design a *Universal Turing machine*, U, which accepts as input a Turing machine and its input, $< M, w >$, and simulates the action of M on the input w.

In fact this is exactly how a computer works, a programming language provides a way of describing an algorithm, which by the Church-Turing thesis is equivalent to a description of a Turing-machine M, at run time the program is supplied with the input w, the computer then simulates M running on w.

We now consider the following language (Sipser p. 161):

$L_{TM} = \{< M, w > \mid M \text{is a Turing machine, and } M \text{ accepts } w\}$.

The Halting Problem Is there a Turing machine which decides L_{TM}?

We note the following:

Theorem L_{TM} is recognizable.

Proof:

Simulate M on w.

If M enters q_{accept} accept.

If M enters q_{reject} reject. $\square$

Thus the problem is whether M halts on every input w.

Theorem L_{TM} is undecidable.

Proof: By Contradiction. (See Sipser p.165)

Suppose that L_{TM} is decidable, thus there is a Turing machine H, which always halts, which recognizes L_{TM}.

We write $H(< M, w >) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$

Now we design a new Turing machine D, which uses H as a 'subroutine'. The input to D is the encoding of a Turing machine $< M >$. On input $< M >$, D runs H on $< M, < M >>$. i.e. H determines the outcome of running the Turing machine M on an encoding of itself. D then reverses the output from H:

$D(< M >) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } < M > \\ \textit{reject} & \text{if } M \text{ accepts } < M > \end{cases}$

We now run D on itself to derive a contradiction:

$D(< D >) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } < D > \\ \textit{reject} & \text{if } D \text{ accepts } < D > \end{cases}$

The statement D accepts $< D >$ indicates that on input $< D >$, D accepts, But it rejects, a contradiction. Thus L_{TM} is undecidable. $\square$

This means that no Turing machine can decide whether a second Turing machine will eventually halt.

Has it crashed, or is it just taking a long time?