Introduction to Computational Complexity: Turing Machines and Diagonalization

Xi Chen

Columbia University

We start with the definition of Turing machines. A Turing machine (TM for short) $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{acc}}, q_{\text{rej}})$ is a 7-tuple, where

1. Q is the set of states;
2. Σ is the input alphabet not containing the special symbol $\sqcup$;
3. Γ is the tape alphabet, with $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$;
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L,R}\}$ is the transition function;
5. q_{start} is the start state;
6. q_{acc} is the accept state; and
7. q_{rej} is the reject state.

How does a Turing machine run? Given an input string $\mathbf{w} = w_1 w_2 \cdots w_n \in \Sigma^*$, initially the tape is set to

$$w_1 w_2 \cdots w_n \sqcup \sqcup \cdots$$

and the head is at the leftmost cell w_1, with state set to q_{start}. Then for each step, the machine moves following the transition function δ. For example, assume the current state is q and the head is over a tape cell containing a symbol a. If $\delta(q, a) = (r, b, L)$, then the machine writes the symbol b into the current cell, replacing the a; goes to state r; and moves left by one cell.

Note that if the head is at the leftmost cell and decides to move left, according to the transition function, then it stays at the same cell for another round. Sometime it is convenient to start by writing a special symbol into the leftmost cell and then shifting the input string to the right by one cell. In this way, we make sure the head knows it is at the leftmost cell (e.g., when we want to move the head back to the beginning of the input string).

Next we define configurations of a TM. Here a configuration is simply a snapshot of a TM. It includes the current state of the TM, the current (full) tape contents, and the current head location. We usually describe a configuration C as follows:

$$C = \mathbf{u}\ q\ \mathbf{v}$$

Here $\mathbf{u}$ and $\mathbf{v}$ are two strings over the tape alphabet Γ and $q \in Q$ is a state. It means that the current tape contents are

$$\mathbf{u}\,\mathbf{v}\,\sqcup\,\sqcup\cdots$$

the current state is q; and the head is at the first symbol of $\mathbf{v}$.

For example, if the current state is q; the tape contents are

$$0110 \sqcup \sqcup 11011 \sqcup \sqcup \cdots$$

and the tape head is at the sixth cell (the second blank symbol $\sqcup$), then its corresponding configuration is $C = \mathbf{u}\ q\ \mathbf{v}$, where

$$\mathbf{u} = 0110 \sqcup \quad \text{and} \quad \mathbf{v} = \sqcup 11011$$

Given two configurations C_1 and C_2, we say C_1 yields C_2 if the TM can legally go from C_1 to C_2 in a single step, following the transition function δ. For example, let $a, b, c \in \Gamma$ and $\mathbf{u}, \mathbf{v} \in \Gamma^*$. Let $q, r \in Q$ denote two states. Then we have

$$C_1 = \mathbf{u}a\ q\ b\mathbf{v} \quad \text{yields} \quad C_2 = \mathbf{u}\ r\ ac\mathbf{v},$$

if $\delta(q, b) = (r, c, L)$.

A Turing machine halts only when it enters one of the two special states q_{acc} and q_{rej}. We call C an accepting configuration if the state is q_{acc}; and a rejecting configuration if the state is q_{rej}. We also say the TM accepts or rejects the input string, respectively.

So one may imagine that there is an infinite directed graph G in which the vertices correspond to configurations and there is a directed edge from C_1 to C_2 if C_1 yields C_2. It is clear that every configuration has outdegree exactly 1 unless it is an accepting or rejecting configuration. Running a Turing machine is then like walking on this graph: Given an input string $\mathbf{w} \in \Sigma^*$, we start at the configuration $q_{\text{start}}\ \mathbf{w}$ and keep following the unique outgoing edge until an accepting or rejecting configuration is reached.

Turing Machines as Algorithms

Given a Turing machine M and a string $\mathbf{w} \in \Sigma^*$, we use $M(\mathbf{w})$ to denote the result of running M on $\mathbf{w}$:

- $M(\mathbf{w}) =$ "yes" if M halts on $\mathbf{w}$ at an accepting configuration;
- $M(\mathbf{w}) =$ "no" if M halts on $\mathbf{w}$ at a rejecting configuraton; and
- $M(\mathbf{w}) = \nearrow$ if M never halts on $\mathbf{w}$.

Given a language $L \subseteq \Sigma^*$, we say a Turing machine M *decides* L if $M(\mathbf{w}) =$ "yes" for any $\mathbf{w} \in L$; and $M(\mathbf{w}) =$ "no" for any $\mathbf{w} \notin L$. In particular, for M to decide a language L, it must halt on any input string $\mathbf{w} \in \Sigma^*$. If such a machine exists, we say L is *decidable*.

Example

Let PALINDROME $\subset \{0,1\}^*$ *denote the following language:* $\mathbf{w} \in$ PALINDROME *iff it reads the same way in both directions. Show that* PALINDROME *is decidable.*

Given a string $\mathbf{w} \in \{0,1\}^*$ of length n, roughly how many steps do your TM take to solve PALINDROME in the worst case (even though we have not formally defined worst case yet)?

If you feel too painful to code a Turing machine yourself, read Chapter 3 of Sipser or Chapter 2 of Papadimitriou! If your TM decides PALINDROME in roughly n^2 steps given any string $\mathbf{w}$ of length n, then congratulations! since you have found essentially the "most efficient" TM for PALINDROME. We will prove that any (single-tape) TM needs $\Omega(n^2)$ steps to decide PALINDROME later in the course when we study communication complexity.

More generally, we can define multitape Turing machines. The only difference is now a TM M may have $k \geq 1$ tapes and heads, though k is still a constant. At the beginning, the input string is stored on the first tape. The transition function now tells us how to operate all the k heads in one move. More formally, we have

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

By $\delta(q, a_1, \ldots, a_k) = (r, b_1, \ldots, b_k, L, R, \ldots, L)$, it means that if the state is q and the ith head is reading a_i, $i \in [k] = \{1, 2, \ldots, k\}$ then the machine should move to state r; the ith head should write b_i into the tape and move left or right as specified.

Similarly, we can define decidability of languages by a multitape Turing machine. The following exercise is easier:

Example

PALINDROME *can be decided by a two-tape Turing machine.*

Can you describe a two-tape TM that decides PALINDROME in $O(n)$ steps given any input string $\mathbf{w} \in \Sigma^*$ of length n?

Are multitape TMs more powerful than single-tape TMs? In particular, is there a language that is decidable by a multitape TM but is undecidable by a single-tape TM? The answer is no.

Theorem

Given any k-tape TM M, we can construct a single-tape TM M' such that $M(\mathbf{w}) = M'(\mathbf{w})$ for any input string $\mathbf{w} \in \Sigma^*$.

The main idea in the simulation is to use a much larger (but still of constant size) tape alphabet, though you can actually avoid this (how?). Assume $\Gamma = \{0, 1, \sqcup\}$ and M has k tapes. The tape alphabet Γ' of the new single-tape TM M' is

$$\Gamma' = \left\{0, \overline{0}, 1, \overline{1}, \sqcup, \overline{\sqcup}\right\}^k$$

so that each cell of M' stores a k-tuple. In this way, the tape of M' can encode k tapes of M, including the k tape heads of M. For example, if the ith cell of M' is $(0, 1, \overline{1}, \overline{\sqcup}, \ldots)$, it means that the ith cells in the k tapes of M are $0, 1, 1, \sqcup, \ldots$ respectively; and the heads of the third and fourth tapes are at the ith cell.

To simulate each step of M, M' needs to make two passes on its tape. In the first pass, M' collects all the cell contents that the k virtual heads of M are reading, i.e., whenever the head of M sees a symbol $\overline{a}$ in the tuple it reads, it remembers which component it is as well as the symbol. For example, if the second component of the current tuple it reads is $\overline{a}$, the M' "remembers" in its state that the head of the second tape of M is reading a. In the second pass, M' follows the transition function δ of M and updates the tape by writing and moving the k virtual heads. To this end, new states need to be introduced in M' to control the head carefully. This way we get a single-tape M that simulates M'. Find in Chapter 2 of Papadimitriou an alternative simulation.

As a corollary, to show a language L is decidable by a single-tape TM, it suffices to construct a multitape TM that decides L. Moreover, if it takes the multitape TM M t steps to halt on an input $\mathbf{w}$, then our simulation described earlier takes $O(t^2)$ steps at most (why?). This quadratic blowup in the number of steps needed to simulate a multitape TM is indeed inevitable, as suggested earlier by PALINDROME.

Next question: Is there a language that is undecidable? Yes. Actually "most" languages are undecidable just by a counting argument. First note that each Turing machine M can be encoded into a finite string over the alphabet $\{0, 1, \#\}$, where we use the special symbol $\#$ as a delimiter. We use $\langle M \rangle$ to denote the encoding of M. As $\{0, 1, \#\}$ is finite, $\{0, 1, \#\}^*$ is countable.

For a language to be decidable, there must be a TM that decides it by definition. Since the set of TMs is countable, the set of decidable languages is countable as well. On the other hand, there is a correspondence between the set of languages $L \subseteq \Sigma^*$, as long as Σ is nonempty, and the set of real numbers. Thus, the set of languages is uncountable.

The fact that the set of real numbers is uncountable (there is no one-to-one correspondence between real numbers and natural numbers) is proved by Cantor in 1873 by a technique called diagonalization. We use it here to show that the following more "concrete" halting problem is undecidable. Let $\Sigma = \{0, 1, \#\}$. Given a Turing machine M and an input string $\mathbf{x}$ (over the input alphabet of M), we use $\langle M, \mathbf{x} \rangle$ to denote an encoding of M and $\mathbf{x}$ in Σ. The halting problem is the following language $L_{\mathsf{TM}} \subset \Sigma^*$: a string is in L_{TM} if it is of the form $\langle M, \mathbf{x} \rangle$ and

$$M(\mathbf{x}) = \text{"yes"}$$

Theorem

The halting problem is undecidable.

Proof.

Assume for contradiction that there is a TM H that decides the halting problem. We use H to construct the following TM D.

1. If the input string is of the form $\langle M \rangle$ do the following
2. Simulate H on input $\langle M, \langle M \rangle \rangle$
3. If H accepts, reject; and if H rejects, accept
4. Otherwise simply reject

Verify that for any Turing machine M, D does not agree with M on $\langle M \rangle$. This includes $M = D$ itself, a contradiction! □