Computability Theory I

Michael Soltys

November 16, 2001

In this part of the course, we will prove that some problems (languages) cannot be solved by any algorithm.

There are problems which do not even have a "bad" algorithm. An example of a problem that cannot be solved by any algorithm is the Halting problem: given a program P and an input x to P, decide whether P halts on input x.

Turing (1936) introduced Turing Machines, and proved that no TM can solve the Halting problem. He also argued that any algorithm can be simulated by a TM; this statement is often called the **Church-Turing Thesis**.

TMs are a very robust model of computation. By this we mean that any "reasonable" modification of TMs does not alter their computing power (this is not the case for Pushdown Automata, where by adding a second stack we make the PDAs more powerful).

For example, we can add any number of tapes to a TM, and this modification results only in a polytime increase in computing speed, but does not allow us to recognize any new languages.

We can also make the tape infinite in only one direction. How can a TM with one-way tape simulate a TM with a two-way tape?

Idea 1: Use a marker for the right end of the used tape space. If scanning the left end and want to shift left, move the entire tape content right one square.

Idea 2: "Fold" the tape. Store the top part of the tape on even squares, bottom part on odd squares.

General Claim: A TM can simulate every reasonable variation of a TM (e.g., a TM with several tapes).

Let $L(M)$ be the language accepted by a TM M. Then $x \in L(M)$ iff M, when started on input x, eventually enters the state q_{accept}. We assume that the input x is in Σ^*, so $L(M) \subseteq \Sigma^*$.

Note that x can fail to be in $L(M)$ for two reasons: either M fails to halt on input x, or M halts in the reject state, q_{reject}.

Definition 1 Let $L \subseteq \Sigma^*$. Then L is *semi-decidable* if $L = L(M)$ for some TM M. (This is also called “recognizable”.) On the other hand, a TM M *decides* L if $L = L(M)$ **and** M **halts for every input** $x \in \Sigma^*$. L is *decidable* if some TM decides L. Note that we are not concerned with the length of the computation for these definitions.

For example, the Halting problem is semi-decidable, but not decidable; we will see later why.

Definition 2 An *enumerator* E is a TM with a separate write-only output tape. E writes strings on its output. $L(E)$ is the set of strings x such that when E starts with a blank tape, x is eventually written on the output. Thus $L(E) \subseteq \Gamma^*$, where Γ is the output alphabet. L is *enumerable* iff $L = L(E)$ for some enumerator E.

Theorem 1 L is enumerable iff L is semi-decidable.

PROOF: The proof basically follows from the definitions of semi-decidability and enumerability, and from the notion of “simulation”:

If L is enumerable, then $L = L(E)$ for some enumerator E, and then $L = L(M)$ where M is a TM defined as follows:

> `On input` x`, ` M `simulates` E `(on a blank tape), and each time` E
> `outputs a` y`, ` M `checks if` $x = y$`, and halts in` $q_{\texttt{accept}}$ `if ‘‘yes’’.`

Thus L is semi-decidable.

Conversely, let M be a TM such that $L(M) = L$. For each string $s \in \Sigma^*$, E must output s only if M accepts s. *Problem*: M may never halt on s. This is why our enumerator E will simulate M on many strings s “in parallel”.

Let $s_1, s_2, \ldots$ be a list of all strings in Σ^*. Then E does the following:

```
for i : 1..∞
    for j : 1..i
        if M accepts s_j within i steps then
           output s_j
        end if
    end for
end for
```

If M accepts $s = s_k$ within l steps, then E will output s when $i = \max\{k, l\}$. Hence, L is enumerable. $\square$

The Halting Problem

We can encode any string over an arbitrary alphabet Σ_1 by using the fixed alphabet $\Sigma = \{0, 1\}$.

For example, let $\Sigma_1 = \{a, b, c\}$. Encode a as 01, b as 011, and c as 0111. Then $aabc$ becomes 01010110111. Note that decoding is unique.

We can also encode the different states of a TM and the transition function δ using the alphabet Σ.

Thus, we can encode arbitrary TMs M and their inputs w by using the alphabet $\Sigma = \{0, 1\}$. Let $\langle M, w\rangle$ denote such an encoding of a TM M and its input w.

Let

$$A_{\mathrm{TM}} = \{\langle M, w\rangle |\ M \text{ is a TM and } M \text{ accepts } w\ \}.$$

Theorem 2 A_{TM} is not decidable.

A_{TM} is however semi-decidable by the *universal* TM U as follows:

> `On input` $\langle M, w\rangle$`, where` M `is a TM and` w `is a string,` U `simulates` M `on input` w`, accepting whenever` M `accepts, and rejecting whenever` M `rejects.`

The TM U behaves as an *interpreter*. The existence of the universal TM was proved by Turing (1936); the proof is not hard, but requires some care.

Diagonalization

The technique used to show the undecidability of A_{TM} was first used by Georg Cantor (1873) to show $\mathbb{R}$ is not countable.

Definition 3 A set A is *countable* if A is finite, or if there is a one-one and onto function $f : \mathbb{N} \to A$.

If A is an infinite countable set, then for some function f, $A = \{f(1), f(2), \ldots\}$.

Example: The set $\mathbb{Q}$ of rationals is countable. Consider an infinite matrix (a_{ij}), where $a_{ij} = \frac{i}{j}$. We can list all the rationals by traversing the diagonals of the matrix, omitting the elements that would cause a repetition.

Theorem 3 (Cantor) The set $\mathbb{R}$ of reals is not countable.

Proof (by contradiction): Suppose $f : \mathbb{N} \to \mathbb{R}$ is one-one and onto. This means that all the reals between 0 and 1 will appear in the following table:

n	the fractional part of $f(n)$
1	$0.\underline{a_1}a_2a_3a_4\ldots$
2	$0.b_1\underline{b_2}b_3b_4\ldots$
3	$0.c_1c_2\underline{c_3}c_4\ldots$
$\vdots$	$\ddots$

We will construct a real number x between 0 and 1 such that $x \neq f(1)$, $x \neq f(2)$, $x \neq f(3)$, and so on. Define $x = 0.x_1x_2x_3x_4...$ so that $x_1 \neq a_1$, $x_2 \neq b_2$, $x_3 \neq c_3$, and so on. Namely, if $a_1 = 1$, then set $x_1 = 2$; if $a_1 \neq 1$, then set $x_1 = 1$. Similarly for b_2 and x_2, c_3 and x_3, and so on.

By construction, $x \neq f(n)$ for every $n \in \mathbb{N}$ since x differs from $f(n)$ in the nth fractional digit. Hence x is not in $\mathbb{R}$, which is a contradiction.

Proof of Theorem 2:

In this proof we are stressing the analogy to Cantor's diagonal argument. Our proof is by contradiction. Suppose that a TM H decides A_{TM}. That is,

$$H(\langle M, w\rangle) = \begin{cases} \text{Accept} & \text{if } M \text{ accepts } w \\ \text{Reject} & \text{if } M \text{ does not accept } w. \end{cases}$$

Let $M_1, M_2, M_3, \ldots$ be a list of all TMs. Consider an infinite matrix (h_{ij}), where $h_{ij} = H(\langle M_i, \langle M_j\rangle\rangle)$. For example, the top left corner of such a matrix could look as follows:

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\cdots$
M_1	<u>accept</u>	reject	reject	
M_2	reject	<u>reject</u>	accept	$\cdots$
M_3	accept	accept	<u>accept</u>	
$\vdots$		$\vdots$		$\ddots$

Assuming the existence of H, we can define a TM D that differs from each M_i. We define D to behave as follows:

> `On input` $\langle M\rangle$`, where` M `is a TM,` D `simulates` H `on` $\langle M, \langle M\rangle\rangle$ `and outputs the opposite of what` H `outputs.`

That is,

$$D(\langle M\rangle) = \begin{cases} \text{Accept} & \text{if } M \text{ does not accept } \langle M\rangle \\ \text{Reject} & \text{if } M \text{ accepts } \langle M\rangle \end{cases}$$

Note that D is a TM since H is. But now consider what D does on its own encoding:

$$D(\langle D\rangle) = \begin{cases} \text{Accept} & \text{if } D \text{ does not accept } \langle D\rangle \\ \text{Reject} & \text{if } D \text{ accepts } \langle D\rangle, \end{cases}$$

which is a contradiction. Hence H cannot exist, and so A_{TM} is undecidable.

Theorem 2 applies to *every* general programming language. That is, the Halting Problem for every general programming language is undecidable.

Finally, consider the following question: Can there exist a TM that makes finitely many mistakes, but otherwise decides A_{TM}?

The answer is no. We can always "patch up" a TM making finitely many mistakes by "hard-coding" the correct answers to the problematic inputs into the finite control of the TM. But then we would obtain a TM that makes no mistakes in deciding A_{TM}, which is impossible by the theorem we just proved.