

Lecture 21: Undecidability, halting and diagonalization

14 April 2009

‘There must be some mistake,’ he said, ‘are you not a greater computer than the Milliard Gargantubrain at Maximegalon which can count all the atoms in a star in a millisecond?’

‘The Milliard Gargantubrain?’ said Deep Thought with unconcealed contempt. ‘A mere abacus - mention it not.’

– The Hitch Hiker’s Guide to the Galaxy, by Douglas Adams.

In this lecture we will discuss the *halting problem* and *diagonalization*. This covers most of Sipser section 4.2.

1 Liar’s Paradox

There’s a widespread fascination with logical paradoxes. For example, in the Deltora Quest novel “The Lake of Tears” (author Emily Rodda), the hero Lief has just incorrectly answered the trick question posed by the giant guardian of a bridge.

“We will play a game to decide which way you will die,” said the man. “You may say one thing, and one thing only. If what you say is true, I will strangle you with my bare hands. If what you say is false, I will cut off your head.”

After some soul-searching, Lief replies “My head will be cut off.” At this point, there’s no way for the giant to make good on his threat, so the spell he’s under melts away, he changes back to his original bird form, and Lief gets to cross the bridge.

The key problem for the giant is that, if he strangles Lief, then Lief’s statement will have been false. But he said he would strangle him only if his statement was true. So that does not work. And cutting off his head does not work any better. So the giant’s algorithm sounded good, but it turned out not to work properly for certain inputs.

A key property of this paradox is that the input (Lief’s reply) duplicates material used in the algorithm. We’ve fed part of the algorithm back into itself.

2 The halting problem

Consider the following language

$$A_{\text{TM}} = \left\{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \right\}.$$

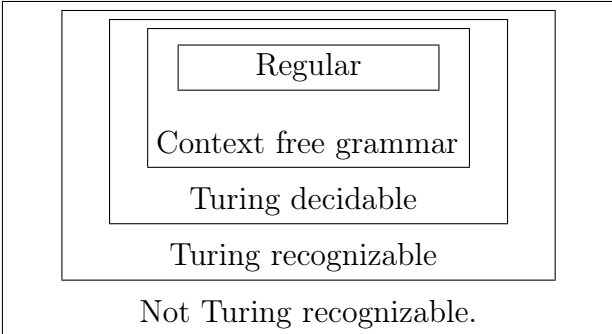
We saw in the previous lecture, that one can build a universal Turing machine U_{TM} that can simulate any Turing machine on any input. As such, using U_{TM} , we have the following TM recognizing A_{TM} :

```

Recognize- $A_{\text{TM}}$ ( $\langle M, w \rangle$ )
  Simulate  $M$  using  $U_{\text{TM}}$  till it halts
  if  $M$  halts and accepts then
    accept
  else
    reject

```

Note, that if M goes into an infinite loop on the input w , then the TM **Recognize- A_{TM}** would run forever. This means that this TM is only a recognizer, not a decider. A decider for this problem would call a halt to simulations that will loop forever. So the question of whether A_{TM} is TM decidable is equivalent to asking whether we can tell if a TM M will halt on input w . Because of this, both versions of this question are typically called the *halting* problem.



We remind the reader that the language hierarchy looks as depicted on the right.

2.1 Implications

So, let us suppose that the Halting problem (i.e., deciding if a word in is in A_{TM}) were decidable. Namely, there is an algorithm that can solves it (for any input). this seems somewhat hard to believe since even humans can not solve this problem (and we still live under the delusion that we are smarter than computers).

If we could decide the Halting problem, then we could build compilers that would automatically prevent programs from going into infinite loops and other very useful debugging tools. We could also solve a variety of hard mathematical problems. For example, consider the following program.

```

Percolate (  $n$ )
  for  $p < q < n$  do
    if  $p$  is prime and  $q$  is prime, and  $p + q = n$  then
      return

  If program reach this point then Stop!!!

Main:
   $n \leftarrow 4$ 
  while true do
    Percolate ( $n$ )
     $n \leftarrow n + 2$ 

```

Does this program stop? We do not know. If it does stop, then the *Strong Goldbach conjecture* is false.

Conjecture 2.1 (Strong Goldbach conjecture.) *Every even integer greater than 2 can be written as a sum of two primes.*

This conjecture is still open and its considered to be one of the major open problems in mathematics. It was stated in a letter on 7 of June 1742, and it is still open. Its seems unlikely that a computer program would be able to solve this, and a larger number of other mathematical conjectures. If A_{TM} is decidable, then we can write a program that would try to generate all possible proofs of a conjecture and verify each proof. Now, if we can decide if a programs stop, then we can discover whether or not a mathematical conjecture is true or not, and this seems extremely unlikely (that a computer would be able to solve all problems in mathematics).

I hope that this informal argument convinces you that its seems extremely unlikely that A_{TM} is TM decidable. Fortunately, we can prove this fact formally.

3 Not all languages are recognizable

Let us show a non-constructive proof that not all languages are Turing recognizable. This is true because there are fewer Turing machines than languages.

Fix an alphabet Σ and define the lexicographic order on Σ^* to be: first order strings by length, within each length put them in dictionary order.

Lexicographic order gives us a mapping from the integers to all strings, e.g. s_1 is the first string in our ordered list, and s_i is the i th string.

The encoding of each Turing machine is a finite-length string. So we can put all Turing machines into an ordered list by sorting their encodings in lexicographic order. Let us call the Turing machines in our list M_1, M_2 , and so forth.

We can make an (infinite) table of how each Turing machine behaves on each input string. This table is depicted on the right. Here, the i th row represents the i th TM M_i , where the j th entry in the row is **acc** if M_i accepts the j th word s_j .

The idea is now to define a language from the table. Consider the language L_{diag} which is the language formed by taking the diagonal of this table.

	s_1	s_2	s_3	s_4	\dots
M_1	acc	acc	rej	rej	\dots
M_2	rej	acc	rej	acc	\dots
M_3	acc	rej	acc	acc	\dots
M_4	rej	acc	rej	rej	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

Formally, the word $s_i \in L_{\text{diag}}$ if and only if M_i accepts the string s_i . Now, consider the complement language $L = \overline{L_{\text{diag}}}$.

This language can not be recognized by an of the Turing machines on the list M_1, M_2, \dots . Indeed, if M_k recognized the language L , then consider s_k . There are two possibilities.

- If M_k accepts s_k then the k th entry in the k th row of this infinite table is **acc**. Which implies in turn that $s_k \notin L$ (since L is the complement language), but then M_k (which recognizes L) must not accept s_k . A contradiction.

- If M_k does not accept s_k then the k th entry in the k th row of this infinite table is **rej**. Which implies in turn that $s_k \in L$ (since L is the complement language), but then M_k (which recognizes L) must accept s_k . A contradiction.

Thus, our assumption that all languages have a TM that recognizes them is false. Let us summarize this very surprising result.

Theorem 3.1 *Not all languages have a TM that recognize them.*

Intuitively, the above claim is a statement above infinities: There are way more languages (essentially, any real number defines a language) than TMs, as the number of TMs is countable (i.e., as numerous as integer numbers). Since the cardinality of real numbers (i.e., \aleph) is strictly larger than the cardinality of integer numbers (i.e., \aleph_0), it follows that there must be an orphan language without a machine recognizing it.

A limitation of the preceding proof is that it does not identify any particular tasks that are not TM recognizable or decidable. Perhaps the problem tasks are only really obscure problems of interest only to mathematicians. Sadly, that is not true.

4 The Halting theorem

We will now show that a particular concrete problem is not TM decidable. This will let us construct particular concrete problems that are not even TM recognizable.

Theorem 4.1 (The halting theorem.) *The language A_{TM} is not TM decidable,*

Proof: Assume A_{TM} is TM decidable, and let **Halt** be this TM deciding A_{TM} . That is, **Halt** is a TM that always halts, and works as follows

$$\mathbf{Halt}(\langle M, w \rangle) = \begin{cases} \text{accept} & M \text{ accepts } w \\ \text{reject} & M \text{ does not accept } w. \end{cases}$$

We will now build a new TM **Flipper**, such that on the input $\langle M \rangle$, it runs **Halt** on the input $\langle M, M \rangle$. If **Halt** $(\langle M, M \rangle)$ accepts than **Flipper** rejects, and if **Halt** $(\langle M, M \rangle)$ rejects than **Flipper** accepts. Formally

<p>Flipper ($\langle M \rangle$) $\text{res} \leftarrow \mathbf{Halt}(\langle M, M \rangle)$ if res is accept then reject else accept</p>
--

The key observation is that **Flipper** *always stops*. Indeed, it uses **Halt** as a subroutine and **Halt** by our assumptions always halts. In particular, we have the following

$$\mathbf{Flipper}(\langle M \rangle) = \begin{cases} \text{reject} & M \text{ accepts } \langle M \rangle \\ \text{accept} & M \text{ does not accept } \langle M \rangle. \end{cases}$$

Flipper is a TM (duh!), and as such it has an encoding $\langle \mathbf{Flipper} \rangle$. Now, consider running **Flipper** on itself. We get the following

$$\mathbf{Flipper}(\langle \mathbf{Flipper} \rangle) = \begin{cases} \text{reject} & \mathbf{Flipper} \text{ accepts } \langle \mathbf{Flipper} \rangle \\ \text{accept} & \mathbf{Flipper} \text{ does not accept } \langle \mathbf{Flipper} \rangle. \end{cases}$$

This is absurd. Ridiculous even! Indeed, if **Flipper** accepts $\langle \mathbf{Flipper} \rangle$, then it rejects it (by the above definition), which is impossible. Indeed, if **Flipper** must reject (note, that **Flipper** always stops!) $\langle \mathbf{Flipper} \rangle$, but then by the above definition it must accept $\langle \mathbf{Flipper} \rangle$, which is also impossible.

Thus, it must be that our assumption that **Halt** exists is false. We conclude that A_{TM} is not TM decidable. ■

Corollary 4.2 *The language A_{TM} is TM recognizable but not TM decidable,*

4.1 Diagonalization view of this proof

Let us redraw the diagonalization table from Section ???. This time, we will include only input strings that happen to be encodings of Turing machines. The table on the right shows the behavior of **Halt** on inputs of the form $\langle M_i, M_j \rangle$. Our constructed TM **Flipper** takes two inputs that are identical $\langle M, M \rangle$ and its output is the opposite of $\mathbf{Halt}(\langle M, M \rangle)$.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$...
M_1	rej	acc	rej	rej	...
M_2	rej	acc	rej	acc	...
M_3	acc	acc	acc	rej	...
M_4	rej	acc	acc	rej	...
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

So it corresponds to the negation of the entries down the diagonal of this table. Again, we essentially argued that there is no row in this infinite table that its entries are the negation of the diagonal. As such, our assumption that **Halt** is a decider, was false.

5 More Implications

From this basic result, we can derive a huge variety of problems that can not be solved. Spinning out these consequences will occupy us for most of the rest of the term.

Theorem 5.1 *There is no C program that reads a C program P and input w, and decides if P “accepts” w.*

The proof of the above theorem is identical to the halting theorem - we just perform our rewriting on the C program.

Also, notice that being able to recognize a language and its complement implies that the language is decidable, as the following theorem testifies.

Theorem 5.2 *A language is TM decidable iff it is TM recognizable and its complement is also TM recognizable.*

Proof: It is obvious that decidability implies that the language and its complement are recognizable. To prove the other direction, assume that L and \bar{L} are both recognizable. Let M and N be Turing machines recognizing them, respectively. Then we can build a decider for L by running M and N in parallel.

Specifically, suppose that w is the string input to M . Simulate both M and N using U_{TM} , but single-step the simulations. Advance each simulation by one step, alternating between the two simulations. Halt when either of the simulations halts, returning the appropriate answer.

If w is in L , then the simulation of M must eventually halt. If w is not in L , then the simulation of N must eventually halt. So our combined simulation must eventually halt and, therefore, it is a decider for L . ■

A quick consequence of this theorem is that:

Theorem 5.3 *The set complement of A_{TM} is not TM recognizable.*

If it were recognizable, then we could build a decider for A_{TM} by Theorem ??.