Lecture 16: Undecidability using diagonalization

16 March 2010

> '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 show that the problem of checking, given an input Turing machine M and a word w, whether M will accept w, is ***undecidable***. We will show this using a proof based on ***diagonalization***. This covers most of Sipser section 4.2.

We first give a two-step proof, and then combine them to give a one-step proof. We think this is less "cryptic" than the proof given in Sipser. (The two-step proof explains clearly the diagonalization; and the one-step proof, though similar to Sipser, doesn't involve feeding a machine's description to itself, etc. as it is not really needed.)

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 Turing machine acceptance problem

Consider the following language

$$\mathrm{A}_{\mathsf{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathsf{TM} \text{ and } M \text{ accepts } w \right\}.$$

In the above, we fix a particular alphabet Σ (say $\Sigma = \{0, 1\}$, and encode all Turing machines M with input alphabet Σ into words over Σ, denoted $\langle M \rangle$. Hence all Turing machines in A_{TM} are over a particular alphabet Σ. Also, by $\langle , M, w \rangle$, where M is a TM (with input alphabet Σ) and $w \in \Sigma^*$, is simply an encoding of the pair $\langle M \rangle$ and w, into a single word over Σ.

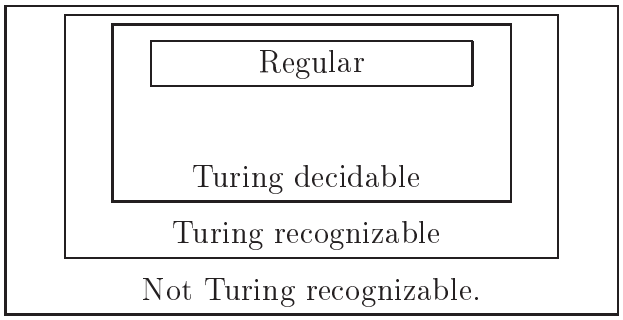
Note that for any TM M (with input alphabet Σ), $\langle M \rangle \in \Sigma^*$, but in general not every string $w \in \Sigma^*$ may correspond to the encoding of a TM.

We saw in the previous lecture, that one can build a universal Turing machine U_{TM} that can *recognize* the above language, by simulating the input Turing machine M on the input word w. Hence, using U_{TM}, we have the following TM recognizing A_{TM}:

Recognize-A_{TM} $(\langle M, w \rangle)$
Simulate M using U_{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 ← 4
        while true do
                Percolate (n)
                n ← n + 2
```

Does this program stops? 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 programs stop, then we can discover whether or not a mathematical conjecture is true or not, and this seems extremely unlikely. We will now prove that A_{TM} is undecidable.

3 A language that is not Turing recognizable

Let us show a 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 ith 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 in depicted on the right. Here, the ith row represents the ith TM M_i, where the jth entry in the row is acc if M_i accepts the jth word s_j.

	s_1	s_2	s_3	s_4	...
M_1	**acc**	acc	¬acc	¬acc	...
M_2	¬acc	**acc**	¬ acc	acc	...
M_3	acc	¬acc	**acc**	acc	...
M_4	¬acc	acc	¬acc	**¬acc**	...
⋮	⋮	⋮	⋮	⋮	⋱

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.

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 cannot be recognized by an of the Turing machines on the list $M_1, M_2, \ldots$. 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 kth entry in the kth row of this infinite table is ¬acc. 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. In particular, $\overline{L_{\text{diag}}}$ is not TM-recognizable (and hence not TM-decidable).*

Intuitively, the above claim is a statement above infinities: There are way more languages (uncountably many) 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 *interesting* 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 Undecidability of Turing-machine membership

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 (Undecidability of Turing-machine membership) *The language A_{TM} is not Turing-decidable.*

Proof: Assume A_{TM} is Turing-decidable. We will show then that $\overline{L_{\text{diag}}}$ (defined above) is decidable, which contradicts Theorem 3.1.

Let M be a Turing machine that decides A_{TM}. Construct the following machine M' that decides $\overline{L_{\mathrm{diag}}}$:

1. Input: w
2. Compute the index of w, i.e. find i such that $s_i = w$.
3. Compute the i'th Turing machine M_i.
4. Feed $\langle M_i, s_i \rangle$ to M.
5. If M accepts, then reject w; if M rejects, accept w.

Since M is a decider for A_{TM}, the Turing machine above is a decider for $\overline{L_{\mathrm{diag}}}$, which contradicts Theorem 3.1. This contradiction proves that our assumption that A_{TM} is Turing-decidable is wrong. Hence A_{TM} must be undecidable.

■

5 Undecidability of Turing-machine membership: Combining the two proofs

We can *combine* the above two proofs to give a simpler (albeit more cryptic) proof that A_{TM} is undecidable.

Theorem 5.1 (Undecidability of Turing-machine membership) *The language* A_{TM} *is not Turing-decidable.*

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

$$\begin{cases} \widehat{M} \text{ accepts } \langle M, w \rangle & \text{if } M \text{ accepts } w \\ \widehat{M} \text{ rejects } \langle M, w \rangle & \text{if } M \text{ does not accept } w. \end{cases}$$

We will now build a new TM **Flipper**, such that on the input w, if $w = s_i$ (i.e. w is the i'th word), runs $\widehat{M}$ on the input $\langle M_i, s_i \rangle$. If $\widehat{M}$ accepts $\langle M_i, s_i \rangle$ then **Flipper** rejects w, and if $\widehat{M}$ rejects $\langle M, s_i \rangle$, then **Flipper** accepts w. Formally

Flipper (w)
  Compute i such that $w = s_i$.
  Compute M_i. res $\leftarrow \widehat{M}(\langle M_i, s_i \rangle)$
  if res is accept **then**
    reject
  else
    accept

The key observation is that **Flipper** *always halts*. Indeed, it uses $\widehat{M}$ as a subroutine and $\widehat{M}$, by our assumptions, always halts as it is a decider. In particular, we have the following: for any $i \in \mathbb{N}$

$$\textbf{Flipper} \text{ on input } s_i : \begin{cases} \text{rejects} & \text{if } M_i \text{ accepts } s_i \\ \text{accepts} & \text{if } M_i \text{ does not accept } s_i. \end{cases}$$

Now **Flipper** is itself a TM (duh!). Let **Flipper** be the j'th Turing machine, i.e. $DHalt = M_j$. Now, consider running **Flipper** on s_j. We get the following

$$\textbf{Flipper}(\text{i.e. } M_j) \text{ on input } s_j : \begin{cases} \text{rejects} & \text{if } M_j \text{ accepts } s_j \\ \text{accepts} & \text{if } M_j \text{ does not accept } s_j. \end{cases}$$

This is absurd. Ridiculous even! Indeed, if **Flipper** (which is M_j) accepts s_j, then it must not accept it (by the above definition), which is impossible. Also, if **Flipper** rejects s_j (note that **Flipper** always stops!), then by the above definition it must accept $\langle$**Flipper**$\rangle$, which is also impossible.

Thus, it must be that our assumption that $\widehat{M}$ exists is false. We conclude that A_{TM} is not TM decidable. ■

Corollary 5.2 *The language* A_{TM} *is TM recognizable but not TM decidable.*

6 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 6.1 *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 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 6.2 *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 $\overline{L}$ 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 6.3 *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 6.2.