Lecture 8:
Recursion Theorem

Valentine Kabanets

October 18, 2016

1 Care with mapping-reductions

In class, we argued that the language $E_{TM} = \{\langle M\rangle \mid L(M) = \emptyset\}$ is undecidable. We proved it by assuming that there is a decider for E_{TM}, and showing how that implies that there is a decider for A_{TM}. A contradiction.

One may be tempted to prove undecidability of E_{TM} by proving the reduction $A_{TM} < E_{TM}$. However, such a reduction is simply not possible! The reason is that the same reduction would also yield that $\bar{A}_{TM} < \bar{E}_{TM}$, implying that the complement of E_{TM} is not semi-decidable. But, the complement of E_{TM} is (essentially) the language $\{\langle M\rangle \mid L(M) \neq \emptyset\}$, which is semi-decidable! Simply enumerate over all strings in lex order, and run M on each in parallel. If M accepts some string, we will get there eventually, and see that $L(M) \neq \emptyset$.

The lesson is: *just because a language is undecidable it doesn't mean you can reduce to it any other undecidable language!* Remember that "$A \leq B$" means both that "B is hard if A is hard" and that "A is easy if B is easy".

2 Recursion Theorem

Consider

$$\text{Self-Print} = \{\langle M\rangle \mid \text{ TM } M \text{ on empty input prints } \langle M\rangle\}.$$

Is Self-Print decidable?

No, it is not. But to prove this, we need the Recursion Theorem.

Theorem 1 (Recursion Theorem)**.** *Let t be any computable function mapping a pair of strings to a string. There is a TM R such that, on input w, TM R outputs the value $t(\langle R\rangle, w)$.*

Intuitively, for any computable function, we can design a TM that will be able to apply that computable function to its own description (e.g., will print its own description).

We'll prove the Recursion Theorem next. We first prove a special case: the existence of a self-printing TM.

2.1 Special case: Self-printing TM

The idea of the proof is to construct a TM that knows its own description. Such a TM, called SELF, will consist of two parts: A and B. Here, A is a TM that outputs $\langle B\rangle$, and such that $\langle A\rangle$

can be reconstructed (computed) from $\langle B \rangle$. Then B is a TM that takes its input (which is its own code $\langle B \rangle$), reconstructs from it the TM A, and outputs $\langle AB \rangle = \langle SELF \rangle$.

For A to be reconstructable from its output $\langle B \rangle$, we define a computable function $q : \Sigma^* \to \Sigma^*$ such that for every $w \in \Sigma^*$, $q(w)$ is the description of a TM that prints out w and halts. Then to reconstruct $\langle A \rangle$, our TM B will simply apply q to its input. Since the input to B will be $\langle B \rangle$, this will yield $q(\langle B \rangle) = \langle A \rangle$.

Thus, to write SELF, we first write B, then write A so that $\langle A \rangle = q(\langle B \rangle)$.

2.1.1 More details

More precisely, we have TM B:

> B: "On input x, compute $a = q(x)$; output a, x."

Define q:

> $q(w)$:= "On input x, output the value of the string w".

Then the TM A is:

> A: "On input x, output the string "On input x, compute $a = q(x)$; output a, x."".

Note that $\langle A \rangle = q($"On input x, compute $a = q(x)$; output a, x."$)$.

The TM AB (where the output of A is the input to B) will print its own code.

2.1.2 Relation to self-printing programs

To relate our TM construction AB of a self-printing TM to the design of self-printing programs, observe that a typical self-printing program has the form

> $a :=$" text ";
> print "$a :=$ value of a in quotes, followed by ';', followed by value of a"

where the "text" is the second line of the program ("print "$a :=$ value ..."").

So, the module A of our self-printing TM construction corresponds to the first line $a :=$"text"; and the module B to the second line (the print command). The module B knows its own code ("text") by looking up the value of the variable a. Then B can also recover the first line of the program (the module A) because the first line is just $a :=$ the value of a in quotes, followed by ';'.

Here's the example of a self-printing program in Python. Note that $chr(i)$ means the ASCII character i. So $chr(97) = a$, $chr(61) ==$, $chr(34) =$ ", and $chr(59) =$;. (The following text should be just one line; it is split into two lines in order to fit the page.)

```
a="b=chr(97)+chr(61)+chr(34);b+=a;print(b+chr(34)+chr(59)+a)";
b=chr(97)+chr(61)+chr(34);b+=a;print(b+chr(34)+chr(59)+a)
```

2.2 General case

To deal with the general case of a function t, let T be the TM that computes t. We construct a new TM as ABT, where A as before will pass onto B the description $y = \langle BT \rangle$ plus the input x to A. Then B, given $\langle x, y \rangle$ from A, will run the function q on $\langle x, y \rangle$ to reconstruct the description $a = \langle A \rangle$. Then B will output ay, x onto T.

Note that T is fixed; the TM for B can be described using the above. Thereafter, we can design the TM A. The resulting TM ABT computes $t(\langle ABT \rangle, x)$, where x is an input to the TM ABT.

3 Applications of the Recursion Theorem

Recall that the Recursion Theorem allows us to design TMs as follows:

> M: "On input w,
>
> 1. Obtain, via the Recursion Theorem, own description $\langle M \rangle$.
> 2. Perform some computation on $\langle M \rangle$ and w."

3.1 Self-Print

As an application, let's prove that Self-Print is undecidable. (That is, virus checking is impossible.) Recall

> Self-Print= $\{\langle M \rangle \mid$ TM M outputs $\langle M \rangle\}$.

Theorem 2. *Self-Print is undecidable.*

Proof. We'll do a reduction from A_{TM}.

Given $\langle M, w \rangle$, construct TM M' as follows.

> M': "On any input,
>
> 1. Obtain own description $\langle M' \rangle$.
> 2. Simulate M on input w.
> 3. If M accepts w, then Print $\langle M' \rangle$ and halt. If M rejects w, then erase all tapes and halt."

It's easy to see that M accepts w iff M' prints $\langle M' \rangle$. Thus, if Self-Print were decidable, we would be able to decide A_{TM}, which is impossible. Hence, Self-Print is undecidable. □

Note that Self-Print is *semi*-decidable: simply simulate a given M, accepting if the simulation ever produces $\langle M \rangle$. It immediately follows that the complement of Self-Print is not semi-decidable.

3.2 Fixed-Point Theorem

Theorem 3 (Fixed-Point Theorem)**.** *For any computable function $t : \Sigma^* \to \Sigma^*$, there is a TM F such that $\langle F \rangle$ and $t(\langle F \rangle)$ are descriptions of two TMs that accept exactly the same language.*

Proof. Construct F as follows.

F : "On input w,

1. Get own description $\langle F \rangle$, via the Recursion Theorem.
2. Compute $t(\langle F \rangle) = \langle G \rangle$ for some TM G.
3. Simulate G on w (accepting iff G accepts)."

Note that $L(F) = L(G)$ since F just simulates G. On the other hand, $\langle G \rangle = t(\langle F \rangle)$. □

3.3 Minimal TMs

As another application, we'll show

Theorem 4. *The language*

$$MIN_{TM} = \{\langle M \rangle \mid \ M \text{ is a minimal TM}\}$$

is not semi-decidable.

Here, a TM M is called *minimal* if there is no TM M' with $|\langle M' \rangle| < |\langle M \rangle|$ such that $L(M') = L(M)$. (That is, of all TMs accepting the language $L(M)$, the TM M has the shortest description.)

Proof. Suppose MIN_{TM} is semi-decidable, say by a TM M. Consider the following TM

C: "On input w,

1. Get own description $\langle C \rangle$ (via the Recursion thm).
2. let $M_1, M_2, \ldots$ be the enumeration of all TMs whose description size is bigger than that of C (i.e., $|\langle M_i \rangle| > |\langle C \rangle|$, for all i).
3. for $i = 1..\infty$,
   simulate M on each $\langle M_1 \rangle, \ldots, \langle M_i \rangle$ for i steps;
   if M accepts $\langle M_k \rangle$ for some k, then go to the next step, else continue with the "for"-loop.
4. simulate M_k on w (accepting if M_k accepts)."

Observe that $|\langle C \rangle|$ is some fixed constant (like 100). Since there are infinitely many distinct computable languages, there are infinitely many distinct minimal TMs. So there are minimal TMs whose description length is bigger than that of C. Hence, within the "for loop" of step 3 of algorithm C, we'll eventually find some TM M_k that is accepted by M (recall that M is an algorithm that semi-decides MIN_{TM}).

Thus, in step 4 of the algorithm for C, we'll have a TM M_k such that

- M_k is a minimal TM, and

- $|\langle M_k \rangle| > |\langle C \rangle|$.

On the other hand, by the construction of C, we have that $L(C) = L(M_k)$. So C is a TM with a shorter description length than M_k, but C accepts exactly the same language as M_k. This contradicts the minimality of M_k. Hence, a TM M cannot exist, and so MIN_{TM} is not semi-decidable. □

4 Summary of the Computability Basics

1. TM as a formalization of the notion of algorithm.
2. Existence of undecidable languages.
3. Diagonalization: a construction of undecidable (even not semi-decidable) languages; A_{TM} is not decidable by a reduction.
4. mapping reductions: can show many languages are "hard".
5. Rice's theorem: any nontrivial property of TMs is undecidable.
6. Recursion Theorem (existence of self-printing programs/viruses; impossibility of perfect virus checkers).