Lecture 6:
Universal Turing machine, and the Halting problem

Valentine Kabanets

September 29, 2016

1 Universal Turing machine

Turing (1936) showed that there exists a machine U that, when given as input (appropriate encoding of) $\langle M, w\rangle$ of a TM M and an input w to M, will simulate M on w (accepting iff M accepts w). Moreover, the running time of the simulation will be at most quadratic in the run time of M on w. (The runtime can be made $t \log t$, where t is the runtime of M, if we allow U to be a 2-tape TM.)

Note that U is a *fixed* TM (with a fixed number of states and tape symbols), which should be able to simulate any TM with any number of tapes and any tape alphabet. To achieve such universal simulation is not trivial, but not hard either. The idea is to encode the states and the tape alphabet of the TM we want to simulate in some fixed alphabet (say, binary), so that, e.g., a single symbol of M would correspond to some constant number of symbols of U. Then we keep track of the configurations of M, and update them according to the program (function δ) of M.

This is similar to the way we constructed a one-tape TM for a given k-tape TM. However, there are important differences which make the construction of U more complicated. Namely, the input machine M can be arbitrary, and so U has to simulate a TM with arbitrary number of tapes, states, and tape symbols. In contrast, the 1-tape TM simulating a particular fixed k-tape TM could "hard-wire" all the information about that fixed k-tape TM.

Nonetheless, the universal TM does exist. This is great since it allows us to have *general-purpose* computers that are capable of running any particular program on the fixed hardware (e.g., your laptop). Thus, we don't need to build a separate piece of hardware to run a particular algorithm!

In my opinion, the proof that a *universal* computational device (universal Turing machine) exists is the most important contribution of Alan Turing! That was the birth of a modern computer.

2 Halting Problem

Can we tell if a given TM is a decider? Or, even simpler, can we tell if a given TM halts on a given input string? No, the latter is the famous Halting Problem that was shown undecidable by Alan Turing in 1936.

Consider the language

$$A_{TM} = \{\langle M, w\rangle \mid \text{ TM } M \text{ accepts input } w\}.$$

Theorem 1. *A_{TM} is semi-decidable.*

Proof. The proof follows from the existence of a *universal* Turing machine. Recall our universal TM U: "On input $\langle M, w\rangle$, simulate TM M on input w, accepting if M accepts w."

We argued that U exists. Obviously, U accepts A_{TM}. $\square$

Theorem 2. *A_{TM} is undecidable.*

Proof. We'll prove it in two stages.

- **Stage 1:** construct a language D such that D is not semi-decidable.
- **Stage 2:** show that if A_{TM} were decidable, then D would be decidable. A contradiction.

Now we provide more details.

Stage 1: Define D using the diagonalization method. Define

$$D = \{\langle M_i\rangle \mid \text{ TM } M_i \text{ does not accept input } \langle M_i\rangle\},$$

where M_i is the ith TM in the complete enumeration of all TMs.

Claim 1. *D is not semi-decidable.*

Proof of Claim: Suppose D is recognizable by some TM M_n. Then consider the question "Is $\langle M_n\rangle$ in $L(M_n)$?"

If the answer is Yes, then we get:

$$\begin{aligned} \langle M_n\rangle \in L(M_n) &\Rightarrow \langle M_n\rangle \notin D && \text{(by the definition of } D\text{)} \\ &\Rightarrow \langle M_n\rangle \notin L(M_n). && \text{(since } D = L(M_n)\text{)} \end{aligned}$$

A contradiction.

If the answer is No, then we get:

$$\begin{aligned} \langle M_n\rangle \notin L(M_n) &\Rightarrow \langle M_n\rangle \in D && \text{(by the definition of } D\text{)} \\ &\Rightarrow \langle M_n\rangle \in L(M_n). && \text{(since } D = L(M_n)\text{)} \end{aligned}$$

A contradiction.

Thus, in both cases we derive a contradiction. Hence, our assumption that D is semi-decidable must be false. $\square$

Stage 2: Suppose A_{TM} is decidable by a TM H. Then we obtain the decider TM for the language D as follows:

> "On input $\langle M_i\rangle$, simulate H on input $\langle M_i, \langle M_i\rangle\rangle$. If H accepts its input, then Reject. Otherwise, Accept."

Since we just proved in Stage 1 that D is not even semi-decidable (let alone decidable), it must be the case that our assumption that A_{TM} is decidable is false. So, A_{TM} is undecidable. $\square$

Thus, we know that there are problems that are semi-decidable but not decidable (e.g., A_{TM} is such a problem). There are also problems that are not even semi-decidable (e.g., language D from the proof above).

3 Types of proofs of undecidability

There are two types of proof for undecidability:

1. diagonalization (e.g., language D)
2. reduction (e.g., A_{TM})

Most of our proofs will be proofs by reduction. We give some examples next time.