**diagonalization language**

The language Ld Which consists of all those strings w such that the Turing machine represented by w does not accept the input w.

Ld = { wi | wi L(Mi)}

**Diagonalization**

Diagonalization is an interesting proof technique that pops up in a lot of problems involving infinite sets.

For example, Cantor used it in 1891 to prove that the set of real numbers is not countable (i.e., that although both the set of integers and the set of real number are infinite, there is a sense in which there are “more” real numbers than integers). Some people think that’s intuitively obvious. Others, particularly those who have seen Hilbert’s paradox of the [infinite hotel](https://en.wikipedia.org/wiki/Hilbert's_paradox_of_the_Grand_Hotel), are surprised that this is true.

The argument goes like this. Suppose that the real numbers are countable – that we can put them into a one-to-one correspondence with the set of integers. In that case, we could write out an infinite list of just the decimal parts of those numbers, in order by their integer equivalents. We will extend each number out to an infinite number of dcimal places, padding with zeroes if necessary. So we might get something like this:

1 .0000000000000...

2 .1000000000000...

3 .3333333333333...

4 .1415926535897...

5 .1231231231231...

⋮ ⋮

We can then prove that there is at least one positive real number, that is not in the list and not associated with any integer. We form that number by plucking the digits from the “diagonal” of the table (.00352….00352…) and addint 1 to each digit, modulo 10 (.11463….11463…).

The resulting number cannot possibly be in our list because, if you were to claim that it is in line jj of the table, it would by definition have a different value in the jthjth digit. Hence there will always be some real numbers left uncounted in any attempt to associate them one-to-one with the integers.

We’re going to use a similar argument to show that a problem exists that cannot be even partially decided by any Turing machine.

**The language**LdLd

Let LdLd be the set of all binary encodings of TMs that do not accept (i.e., that fail or never halt on) their own encoding when presented it as an input.

OK, it’s a bit odd. But still, a binary encoding is just a string, and any set of strings is a language.

Is there a TM that accepts LdLd?

* This question gives us two “levels” of TM. There are the TMs that are in the encodings and now another that decides wither they belong in this set of strings.
  + If the idea of a TM that decides problems about other TMs seems strange, ask yourself what programs like compilers or debuggers are, if not programs that operate on other programs?

Now, let’s make a table. On each axis of the table we will list all TMs, encoded as described above, in some order. The very existence of our encoding means that the set of all TMs is countable, because we have mapped them onto binary integers.

In the (i,j) entry of the table, we put a 0 if TMjTMj accepts the binary encoding of TMiTMi as input.

*Question:* Is there a TM somewhere in that list that accepts LdLd?

*Answer:* No, there can’t be.

Consider the diagonal elements of that table. Those (i,i) elements describe the TMs that accept themselves (i.e., their own numeric encoding) as input.

Now form a string DD as the *complement* of the diagonal elements. The elements of this string with 1’s would then be the strings/TMs in LdLd.

Is there a TM, somewhere in this table, that computes DD? Suppose we believed that TMkTMk computed DD. The row kk of the table would be DD. But we know that we formed DkDk by taking the complement of element (k,k)k,k), so they can’t possibly be equal. Hence there cannot be any TM in the table that has DD as its row.

Therefore there cannot be a TM that computes D. Therefore there is no TM that accepts LdLd.

This is an example of a problem that is not even partially decidable.