Something that we have yet to formally address is what exactly is computation / an algorithm. Informally, computation is the execution of a set of instructions, i.e. an algorithm.

We've claimed that a TM is an abstract model capable of computing everything computable. We will not prove this is the case, as it's beyond the scope of this class, but we will mention that the Church-Turing Thesis provides a formal definition of an algorithm and argues the case for TMs (and λ-calculus).

This does leave us with the question of what is the right level of detail to describe an algorithm? The obvious answer is to provide a full formal TM. However, TMs are cumbersom and rarely enlightening.

At a step up, we may describe how a TM is to be implemented via an English description. This is largely the approach we have taken so far for our first steps.

At a higher level still, we give English prose with no regard to implementation details. So long as we don't try to peform uncomputable operations, there is always some way to convert this prose into a TM, the exact details of which rarely concern us.

$\langle \cdot \rangle$ here means we encode p into a string; how doesn't really matter

<u>Ex 1</u> $L = \{\langle p \rangle \mid p$ is a polynomial with rational coefficients and an integral root$\}$

We give a TM M for which $L(M) = L$.

```
M = " On input <p>, where p is a polynomial of k variables and
      rational coefficients,
   1) Fix a computable enumeration $\phi$ of $\mathbb{Z}^k$
   2) For i = 1 to $\infty$
        a) If $p(\phi(i)) = 0$, accept "
```

Notice that this algorithm says nothing about <u>how</u> we implement this algorithm (or indeed what model of computation we should use).
Nonetheless, each step is computable. $\mathbb{Z}^k$ is countable, and thus ϕ is computable.
Therefore there is an ennumerator for it
More over, p is a finite object and can thus be evaluated.
We do, however, stipulate that p has coefficients that are finitely specified (in $\mathbb{Q}$). For p to have arbitrary coefficients makes things a great deal more complicated (see oracles).

Regardless, if p has an integral root, M will find it eventually. If p does not, then M will loop forever.
Thus $L(M) = L$.

Here M <u>recognizes</u> L. Can we give an N that <u>decides</u> L? Or an $\overline{M}$ such that $L(\overline{M}) = \overline{L}$, which together with M would constitute L?

It turns out no. There are some nasty polynomials that make a general yes/no impossible to give (at least without an oracle).

A natural next question to ask is how do we know if a language is decideable? How do we know when it's not?

There are several techniques to answer these questions, which we will talk about next, but to conclude this topic, we will first discuss an incredibly useful and (way back when) surprising theorem.

$\boxed{\text{Thm}}$ There is a universal TM U which, given any TM M and any string w as input, will simulate $M(w)$.

In other words if we have $U(\langle M,w\rangle)$ accept/reject according to $M(w)$, then $L(U) = \{\langle M,w\rangle \mid M \text{ is a TM which accepts } w\}$. We will late prove that this language is recognizable but not decidable.

We omit the proof of this theorem here as it amounts to "do what M tells you to do" and is similar to proofs we did with TM variants.