Lecture 24: Limits of Computation

MATH230

Te Kura Pāngarau | School of Mathematics and Statistics
Te Whare Wānanga o Waitaha | University of Canterbury

Outline

(In)Completeness v. (Un)Decidability

Gödel showed that there are sentences α in PA for which neither α nor $\neg\alpha$ can be proved from the axioms of PA.

This still leaves room for there to be an effective procedure that could determine whether an arbitrary formula is a theorem of a particular first-order theory. In the case of Gödelian sentences it would just return No for both α and $\neg\alpha$.

- Entscheidungsproblem
- Decidability of arithmetic
- Decidability of a first-order theory $\mathcal{T}$

There are many decidability problems. Theoremhood for any first-order theory.

With some concrete notion of effective procedure at hand, mathematicians were in a position to determine whether some theories were decidable.

Church-Turing Thesis

This is an agreement/consensus among mathematicians and computer scientists that we can swap “effective procedure” (“algorithm” or “mechanical procedure”) for “computable by a Turing machine” in any sentence. This is *not* a theorem.

Two sections of Turing’s paper introducing Turing machines are dedicated to philosophical arguments for why Turing machines capture what we mean by computation.

Furthermore, we can swap “computable by a Turing machine” to “computable according to $\langle\langle$any model equivalent to Turing machines$\rangle\rangle$” in any such sentence.

Sometimes referred to as the Church-Turing-Kleene thesis to account for the first three models of computation that were proven to be equivalent.

Gödel Numbering, again!

Turing uses the standard form of the instructions to encode every Turing machine to a number. One further change he makes is that symbols must be of the form

$$S_0 = b, S_1 = 0, S_2 = 1, \dots, S_n$$

For example: q_1, S_0, S_1, R, q_2

Writing the instructions as a string, separated by semicolons, he made the following substitutions.

- $q_i \mapsto$ D followed by i As.
- $S_j \mapsto$ D followed by j Cs.

The strings describing a Turing machine are written using the symbols $A\ C\ D\ L\ R\ N$ and ;

Universal Turing Machine

With Turing machines compiled down to an integer, we can then load them onto the tape for another Turing machine.

In fact, Turing went to great pains to describe a machine, $\mathcal{U}$, that could read the description number of any machine, read an input tape, and then mimic the output of the input machine on that input tape.

It is not at all as clean as you might hope, there are many messy details, but the output at any given step will, eventually, be printed, somewhere, on the tape with the initial description number.

Previously we'd need one primitive recursive function for each procedure, or setup one λ-expression for each procedure. Turing showed that, at least in theory, there is just one machine that can be *programmed* to carry out *any* computation.

Circle Free Machines

Machines that go on printing (in F-squares) forever are called circle free machines. Otherwise, a machine is called circular.

Can we decide (i.e. with a TM) whether any given Turing machine is circle free?

Turing proves that there can be no Turing machine that would decide whether any Turing machine is circle free. His is a proof by contradiction.

For sake of that contradiction, then, let us assume $\mathcal{D}$ is a Turing machine that decides whether an input machine is circle-free.

Circle Free? Undecidable!

Turing embeds this hypothetical machine $\mathcal{D}$ inside another Turing machine, $\mathcal{H}$, which he describes as executing the following loop:

1. Generate the next integer n,
2. Test whether integer is the D.N. of a TM $M(n)$
3. Use $\mathcal{D}$ to test if circle free
    1. If circle free, then (using $\mathcal{U}$) print nth element of $M(n)$,
    2. else, print previous value on tape
4. return to the top of the loop.

Turing asks: What will H do when it arrives at the description number of itself? Is H circle free?

Halting Problem

This argument has been adjusted over time to the following which you can find all over the internet. Mark Jago has a good explanation on his website and on Computerphile.

Print 0? Undecidable!

Is there a Turing machine, $\mathcal{E}$, that can tell whether any input Turing machine ever prints 0? Choice of symbol not important!

Turing argues: if there is such a machine, then we could design another machine which would tell us whether any input machine prints *infinitely* many 0s.

But this knowledge could be used to say whether the machine is circle-free. Thus, there can be no such machine to tell whether any input machine ever prints 0.

Entscheidungsproblem

Next Turing makes his final move toward deciding, in the negative, the Entscheidungsproblem. He writes down a predicate, in a first-order language, that can be interpreted (in the correct model) as saying

$$P(n) : \text{Turing machine M(n) will print 0}$$

If there were a Turing machine that could decide theoremhood of *any* well-formed formula, then it would be able to decide whether this is a theorem for each n and hence decide if any machine would ever print 0.

Thus such a machine could solve a problem he'd already shown to be uncomputable. Turing reduced the Entscheidungsproblem to the halting problem.

Peano Arithmetic

Peano arithmetic is strong enough to talk about the natural numbers.

Turing showed that Turing machines can be encoded with numbers.

Using this fact means you can analyse Turing machines *within* Peano arithmetic.

Much of the above proof of the entscheidungsproblem thus carries over to prove the undecidability of Peano arithmetic.

This was figured out by others, after the publication of Turing's paper.

Hilbert's ~~Dream~~ Nightmare

Therefore Peano arithmetic is (i) incomplete (ii) undecidable, and (iii) can't be proven to be consistent using finite means.

Hilbert presented his dream about metamathematics in 1921.

Presburger and Skolem made *positive* ground $\sim$ 1929.

Together Gödel and Turing - with the help of many others - turned those dreams into a nightmare by 1936.

What did they actually prove about mathematics?

What does this mean for mathematicians?

What does this mean for computer scientists? Programmers?

Further Reading

Here are some recommended reading to follow up on the lecture content. They are all freely available online.

- Stanford Encyclopedia Articles:
  1. Turing machines
  2. Church-Turing thesis
  3. Computability and complexity theory