CS 245: TUT 105 - Tutorial 12

Last time:
- Turing Machines
  - ↳ How to run them on inputs
  - ↳ How to construct them

This time:
- Undecidability (reduction to Halting Problem)
- Gödel's Incompleteness Thm.

Meme(s) of the week:

Review:

Last time, we introduced the study of Turing Machines as a model for computation, as well as the notion of *computable function*, a function from a set of strings over a finite alphabet to itself whose outputs can be computed in finite time by executing a Turing Machine on its inputs.

In the lecture notes, it is discussed how strings over a finite alphabet are capable of encoding many (in fact, pretty much *any*) mathematical objects:

- Integers
- Rational numbers
- Vectors
- Matrices
- Graphs
- Functions (on finite domains).

Some other examples (mine):

- Algebraically defined real numbers (by encoding their minimal polynomials).
- Standard functions of calculus (by adding function symbols for sin, cos, tangent, sqrt, etc. as well as standard field ops)
- Functions which are solutions of differential equations (by encoding the differential operator that they satisfy, as well as the right-hand-side and enough initial conditions to specify the solution uniquely).
- Sequences (by encoding their recurrence relations, as well as enough initial conditions to specify them uniquely).
- Groups (by encoding their *group presentations*, which are a way of representing a group by a finite collection of generators and relation)

The Biggest Example: Turing Machines!

TMs themselves can be represented as strings, by encoding their finite list of states, the start and accepting states, as well as their transition functions and read/write alphabets, formatted to be interpretable by another TM.

It turns out, there exists (theoretically) a Universal Turing Machine — one which simulates any TM M on an input x. (Modern general-purpose computers are basically this).

> **Theorem 1.** There is a **universal Turing machine** U that for each input $\langle M \rangle, x$ simulates the Turing machine M on input x in the following sense:
>
> - If M halts on input x, then U halts as well and outputs the same string.
> - If M does not halt on input x, then U does not halt either.

In the lecture notes, it was shown (directly) that certain functions which take Turing Machine string encodings as input are uncomfortable, such as:

$$\text{SelfReject}(\langle M \rangle) = \begin{cases} 1, & \text{if } M \text{ outputs } 0 \text{ on input } \langle M \rangle \\ 0, & \text{o/w} \end{cases}$$

$$\text{Halting}(\langle M \rangle, x) = \begin{cases} 1, & \text{if } M \text{ halts on input } x \\ 0, & \text{o/w} \end{cases}.$$

***The Halting function being uncomputable is *very* important, because we can use it to show that many other functions are uncomputable, hence many decision problems are undecidable.

The problem of *deciding* (outputting a YES/NO answer in finite time, using an algorithm) whether a given Turing machine M halts on a given input x is called the **Halting Problem.**

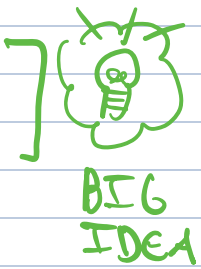
What we've shown here is that the Halting Problem is *undecidable*, i.e. no Turing machine (and hence, no algorithm, if you believe the Church Turing Thesis) can decide it.

Gödel's Inc. Th.:

We would like sets of axioms to satisfy 4 properties:

- Computability: A TM should can distinguish axioms from all other sentences.
- Consistency: We cannot derive a contradiction from our set of axioms.
- Soundness: Every sentence that can be proven (formally) from our axioms should be true in the intended model.
- Completeness: For every sentence, we should be able to provide a formal proof of it or its complement from the axioms.

What Gödel showed (and we will state) is that for "rich enough" mathematical theories, it is impossible to satisfy all 4 of these properties simultaneously.



By "rich enough", we mean that a theory is capable of expressing properties of Turing Machines. Formally…

Def.: A set of axioms Γ is **TM-expressive** if there is a computable function Encode: $(\langle M \rangle x) \longmapsto \varphi_{M,x}$ producing a sentence $\varphi_{M,x}$ so that

$$\Gamma \vdash \varphi_{M,x} \iff M \text{ halts on } x.$$

It can be shown that many of the axiom sets that mathematicians (implicitly, perhaps even unknowingly) use everyday in their work, are TM-expressible.

Lemma: The set of Peano Axioms is TM-expressive. As a result, any axiom system that is sufficiently expressive to encode statements about Peano arithmetic (including the ZFC axioms of set theory) is also TM-expressive.

Thus, Gödel showed that for such systems, it's impossible to satisfy all 4 properties:

Theorem: (Gödel's First Incompleteness, initial version):
Every set of axioms that is TM-expressive, computable, and sound must be incomplete.

Pf.: See notes.

Even worse, we can drop soundness and replace it with the weaker notion of consistency and the result still holds:

Theorem: (Gödel's First Incompleteness Theorem):
Every set of axioms that is TM-expressive, computable, and consistent must be incomplete.

Pf.: See notes.

Moreover, Gödel constructs a sentence that *actually witnesses* the theorem… i.e. a sentence for which we cannot obtain a formal proof of it or its negation.

Thm.: (Gödel's 2nd Incompleteness Thm): Such a sentence is…:

$\neg \varphi_{D,\langle D \rangle}$:= "A universal TM D doesn't halt on its own input."

~Fin.~

Practice Problems:

1. Gödel's Completeness vs. Incompleteness Thms.

What is the difference between these two results? Can you state them both? Can you describe what they mean?

Solution:

Gödel's Completeness theorem:

- Says that if a sentence is true in EVERY model of \Gamma, then there's a formal proof of it.

Gödel's Incompleteness Theorem:

- Says that we can't replace "every model of \Gamma" with "the one model we care about, like the natural numbers", no matter how we choose \Gamma.

2. Examples of Undecidability / Uncomputability:

One of the most important uses of the results of this course is showing that certain problems are *undecidable*… the best tool for doing this is usually NOT a direct proof that the corresponding function is uncomputable.

Instead, you should try and reduce the decision problem to a problem that you already know is undecidable, like the Halting problem.

*****In other words (using the language of this course), to show a function is uncomputable, your best bet is often to show that if we could compute it, then we could compute Halting(<M>,x), which is obviously not the case!**

(a.) Consider the exists-halting-input problem below.

> Given an arbitrary program P, does there exist an input I such that P halts when run with input I?

Prove that the exists-halting-input problem is undecidable.

* I.e., prove that the program

$$\text{ExistsHalting}(\langle M \rangle) \mapsto \begin{cases} 1, & \text{if } \exists w : M(w) \text{ halts} \\ 0, & \text{o/w} \end{cases}.$$

Solution: AFSOC ExistsHalting is computable. With our blackbox algorithm for computing it in-hand, we construct the following algo (I.e. "Turing Machine") which computes Halting:

```
Halting(<M>, w):
  • Build the following program (as a string):
      P(x):                  ← Takes any input.
        • Run M on w.        ← Runs M on w.
  • return ExistsHalting(P).
```

Proof that this computes Halting: Let (M,w) be a TM-input pair. Two cases to consider:

1. M halts on w: Then, no matter what input x is passed into P, P(x) will halt, by its definition. Thus ExistsHalting(P) returns 1.
2. M doesn't halt on w: Then, no matter the input x to P, P(x) won't halt, thus ExistsHalting(P) returns 0.

Thus, the algorithm Halting computes the function Halting(<M>w), meaning Halting(<M>w) is computable. But this is absurd, since we know it's uncomputable.

Thus our original assumption is false; ExistsHalting(<M>) is uncomputable. □

(b.) Assume that the following problem, denoted by EMPTY_{TM} is undecidable:

"EMPTY_{TM}: Given a Turing Machine M, is L(M) = ∅?"

The language of all strings accepted by the TM.

(L(M) denotes the language accepted by the Turing machine M, and is defined as the set of all input words w on which Turing machine halts in an accepting/final state.)

Use this fact to show that the following problem, denoted by EQ_{TM}, is undecidable:

"EQ_{TM}: Given two Turing machines M_1 and M_2, is $L(M_1) = L(M_2)$?"

✱ I.e., show that $EQ_{TM}(\langle M_1\rangle, \langle M_2\rangle) = \begin{cases} 1, & \text{if } L(M_1)=L(M_2) \\ 0, & \text{o/w} \end{cases}$ is uncomputable.

Solution: Let's AFSOC that EQ_TM is computable, and suppose we have an algo EQ_TM(<M_1>,<M_2>) for computing it in hand.

We now construct an algo ("TM") computing EMPTY_TM:

$EMPTY_{TM}(\langle M\rangle)$:

- Build the following program (as a string):

  $M_{no}(x)$:
  return 0.

  Takes any input rets. 0.

- If ($EQ_{TM}(\langle M\rangle, \langle M_{no}\rangle) = 1$):
  return 1
  else:
  return 0

Proof that this computes EMPTY: First, observe that that TM M_{no} has empty language, i.e. accepts nothing. Now, to decide EMPTY_TM, two cases to consider:

1. Suppose M doesn't accept anything, i.e. L(M) is empty. Then, its language equals L(M_no), hence EQ_TM(<M>,<M_no>) returns 1, and so our algo does too.
2. Suppose L(M) is nonempty: Then, the languages of M and M_no differ, so EQ_TM(<M>,<M_no>) returns 0, hence so do we.

Thus we've shown EMPTY_TM is computable, contradicting the standing assumption for this problem.

✱ BONUS: (3.) Show that the EMPTY_TM is uncomputable. (Hint: Try reducing to Halting.)

~ Fin. ~