CS 4510 Automata and Complexity March 4th 2024

Lecture 14: Incompleteness and Undecidability

Lecturer: Abrahim Ladha *Scribe(s): Yitong Li*

1 Solid Foundations

We refresh where we left off from last time. Frege wrote Begriffsschrift, a first attempt at a modern system of logic. These systems attempt to model pure thought itself. Using a formal language, they remove ambiguity from our intutitive way of thinking, and deduction is done by rules of symbolic manipulation. Russell showed his paradox within the system of Frege. Using the generality of the axiom of unrestricted comprehension, he was able to induce a fragility, and derive an inconsistency. Russell and Whitehead then spent the next 12 or so years writing Principia Mathematica. It is a many volume text which seeks to finish what Frege could not. By using a theory of types, it hoped to remove the power of self-reference. Russell and Whitehead were parts of the schools of thought which sought to build a logical foundation for all of mathematics. They wanted to produce a set of axioms such that all theorems and truths derivable in mathematics could be taken as theorems of logic.

1.1 Proving the Strength of the Foundation

You write three thousand pages of axioms and theorems. If your foundation does actually secure all of mathematics, you should prove the following two properties:

Definition 1.1 (Consistency)**.** A set of axioms F is consistent if $F \nvdash (0 = 1)$.

Consistency is the bare minimum requirement for a set of axioms to be useful. A useful theory cannot contain cognitive dissonance. Another way to phrase consistency is that for p any statement, $\forall p[\ p \wedge \neg p\ ]$ is false. Let $Con(PM)$ be the statement which asserts that Principia Mathematica was consistent. If Principia really is a foundation for all of mathematics, then Russell and Whitehead were trying to prove the consistency from within the system itself, $PM \vdash Con(PM)$.

Definition 1.2 (Completeness)**.** A set of axioms F is complete if for any true theorem T, there exists a proof of T from within the axioms.

Theorems can be true and provable. Any theorem which is provable is necessarily true, as there exists a demonstration of its truth. So provable $\implies$ true. In order to show that a set of axioms is complete, it is necessary to show that every true theorem has a proof. This concludes that true $\iff$ provable. There cannot exist any statement which is true but not provable.

Russell and Whitehead (and others) would go on to spend many many years toiling away to show that Principia Mathematica could provably serve as a consistent and complete axiomatic system; a foundation for all of mathematics.

2 Gödel Incompleteness

Godel showed the futility of Russell and Whitehead's effort. He noted that although the theory of types seemed to prevent a sentence referencing its own truth, it did not discuss a sentence referencing its own provability. In fact, nothing could prevent this.

Theorem 1 (Gödel's First Incompleteness Theorem)**.** Any Axiomatic system capable of sufficient arithmetic cannot be both consistent and complete.

By sufficient arithmetic, we mean only the most elementary operations, such as addition and multiplication. There are restricted toy theories which may be consistent and complete.

2.1 Gödel Numberings

In order to discuss metamathematics, Gödel has to be able to discuss these properties from within a system of sets, types and numbers.

Definition 2.1 (Gödel Numberings)**.** A Gödel numbering is a function $\Gamma : \Sigma^* \to \mathbb{N}$. It is an injective function such that to each string, especially the well formed formulas, there corresponds a unique number.

The formal language of the axiomatic system consists of a finite set of primitive symbols. To each primitive symbol, we assign the following numbers.

x	$T(x)$
0	1
S	3
$\neg$	5
$\vee$	7
$\forall$	9
(	11
)	13

Notice we do not need all symbols. Recall that by DeMorgans laws $\exists = \neg\forall$, and $\neg\vee = \wedge$. For variables, we will assign primes greater than 13.

x	$T(x)$
x	17
y	19
...	...

For f a well formed formula, it can be treated as a sequence of symbols over the alphabet of the formal language. We can compute $\Gamma(f)$ by first interpreting f as a string of symbols, and converting to an ordered set of numbers, from the table. Then for our ordered set, we compute them as a product of prime powers. Let $f = f_1...f_n$. Then let the number for symbol x_i be $T(f_i)$. We define

$$\Gamma(f) = \prod_{i=1}^{n} p_i^{T(f_i)} \tag{1}$$

Where p_i is the i'th prime. Lets do an example. Consider the law of excluded middle. As a logical statement it says every statement is either true or not true.

$$\forall x(x \vee \neg x)$$
$$\forall, x, (, x, \vee, \neg, x,)$$
$$9, 17, 11, 17, 7, 5, 17, 13$$
$$2^9, 3^{17}, 5^{11}, 7^{17}, 11^7, 13^5, 17^{17}, 19^{13}$$
$$2^9 \cdot 3^{17} \cdot 5^{11} \cdot 7^{17} \cdot 11^7 \cdot 13^5 \cdot 17^{17} \cdot 19^{13}$$
$$1890436320097715682938344341799980485702396688621865313255411112629182022 25 \cdot 10^9$$

The important part is not the number itself, but the injectivity of Γ. No formulas may map to the same number, and this property is inherited from the fundamental theorem of arithmetic. The fact that every number admits a unique prime factorization.

The representation of other objects as number was new for its time, but it is not new to us. We regularly deal with all kinds of things represented in binary, as this is how computers must interact with them. You should think of f as an object of the system, such as a formula or statement. Think of $\Gamma(f)$ as a description of the object, an encoding. There are of course, multiple equivalent encodings.

By encoding meta-mathematical objects as numbers, Gödel has the ability to discuss meta-mathematical properties using number theory. For example, Suppose it is true that for some number $\Gamma(f)$ and some i, that $p_i^5 p_{i+1}^5 | \Gamma(f)$. This implies that in the representation of f, there exists the two symbols in sequence "¬¬". An axiom may be applied to simplify f, and transform its Gödel number appropriately. Following this way, he develops forty-five different number theoretic relations and formulae. In order to allow a theory of sets and numbers to discuss properties of statements and formulas, Gödel basically has to engineer up an entire computer. Important to us are the following two relations. We simply define them and do not elaborate too much on their construction from primitive arithmetic.

Definition 2.2 (Demonstration)**.**

$$\forall x \forall y[x \textbf{ Dem } y] \iff \Gamma^{-1}(x) \text{ is a proof of } \Gamma^{-1}(y)$$

x is a Gödel numbering of some proof of a statement which y is a numbering of. $\Gamma^{-1}(x)$ is a proof of $\Gamma^{-1}(y)$.

These relations are defined only on numbers but discuss properties of the objects that the numbers encode. The construction of this relation is quite complicated, but it just checks the correctness of the proof of $\Gamma^{-1}(x)$, that each step is an axiom or a correctly applied deduction from the axioms, and that the resulting truth is $\Gamma^{-1}(y)$.

Definition 2.3 (Substitution)**.**

$$\forall x \forall v \forall y[x \textbf{ Sub } (v, y)]$$

Where v is a free variable of $\Gamma^{-1}(x)$,$[x \textbf{ Sub } (v, y)]$ substitutes v in x with the number y.

This formula takes the Gödel numbering x of some formula $\Gamma^{-1}(x)$, and if there is a free variable v, it evaluates it at the number y. If y is a numbering of something else, this will perform composition. To the level of arithmetic, this works by dividing out and multiplying in the appropriate prime powers, as well as some additional work.

2.2 Gödel's First Incompleteness Theorem

Gödel's First Incompleteness Theorem. Consider then the formula with one free variable:

$$f(x) = \neg\exists p\ [p\ \mathbf{Dem}\ (x\ \mathbf{Sub}\ (17, x))]$$

There does not exist a proof p where p proves x substituted for each instance of 17 with number x. Here, f is a formula like any other, and thus has a Gödel numbering, $\Gamma(f)$. What happens if we evaluate the formula f at the number $\Gamma(f)$? Let us compute $f(\Gamma(f))$.

$$f(\Gamma(f)) = \neg\exists p\ [p\ \mathbf{Dem}\ (\Gamma(f)\ \mathbf{Sub}\ (17, \Gamma(f)))]$$

It is quite messy. Let us simplify the inside portion of what is being demonstrated.

$$\Gamma(f)\ \mathbf{Sub}\ (17, \Gamma(f))$$

Here the description of f is a function of x. Recall that for $x\ \mathbf{Sub}\ (v, y)$, anywhere in the numbering x, we replace free variables v with the values y. So we may understand this to mean that we take the numbering of f, and anywhere there exists a free variable of $x = 17$, we replace it with the number $\Gamma(f)$. That is simply the same thing as evaluation of the function $f(\Gamma(f))$!

$$\begin{aligned} \Gamma(f)\ \mathbf{Sub}\ (17, \Gamma(f)) = \\ \neg\exists p\ [p\ \mathbf{Dem}\ (\Gamma(f)\ \mathbf{Sub}\ (17, \Gamma(f)))] = \\ f(\Gamma(f)) \end{aligned}$$

Let us replace this into the original

$$f(\Gamma(f)) = \neg\exists p\ [p\ \mathbf{Dem}\ f(\Gamma(f))]$$

and for simplicity, lets say $f(\Gamma(f)) = g$

$$g = \neg\exists p\ [p\ \mathbf{Dem}\ g]$$

In words, this says "I am not provable" or "There does not exist a proof of me". Assume to the contrary that the system we are working within is both complete and consistent. Since g has no free variables, it must be either true or false. We have two cases:

- If g is true, then g asserts that there is no proof of g, so we have a true statement which is unprovable. The existence of a true but unprovable statement implies that the system we are working in is incomplete, a contradiction.
- If g is false, then by our consistency and completeness assumptions, $\neg g$ is provably true. What is $\neg g$ as a statement?

  $$\neg g = \neg\neg\exists p\ [p\ \mathbf{Dem}\ g] = \exists p\ [p\ \mathbf{Dem}\ g]$$

  By cancellation of double negatives, we see that $\neg g$ asserts that there exists a proof of g. If there exists a proof of g, then g is true. So we see that both g and $\neg g$ are true. Since $[g \wedge \neg g]$ is true, we are inconsistent. Again, a contradiction.

No such system capable of expressing sufficient arithmetic can be both consistent and complete. □

This is a proof by diagonalization. We performed a negated self-reference. While Russell's paradox was able to construct a sentence which said "I am not true". Gödel's sentence was "I am not provable".

2.3 Gödel's Second Incompleteness Theorem

Not only does Gödel prove that achieving a complete and consistent axiomatic system with sufficient arithmetic is impossible, but any system is incapable of proving its own consistency.

Theorem 2 (Gödel's Second Incompleteness Theorem)**.** Let $Con(F)$ be the statement asserting the consistency of the axiomatic system F. If F is a formalized system with sufficient arithmetic, then

$$F \nvdash Con(F)$$

The consistency of F cannot be proven from within F.

Proof. Assume to the contrary $F \vdash Con(F)$. That there exists a proof of the consistency of F from within F. Let this proof be denoted as C. Since the proof of Gödel's first incompleteness theorem assumes (to the contrary) the consistency of F, we may replace this assumption with the proof C. Then we proceed and observe $C \implies g$, our diagonal sentence. Since we can construct g, then F was not simultaneously consistent and complete, a contradiction. Therefore, no proof C of the consistency $Con(F)$ can be proved from within F. No system is capable of proving its own consistency. □

It turns out that some toy systems can be complete and consistent, but they cannot prove their own consistency. You need to use techniques from outside the system to prove them. If Principia was attempting to be a model for all of mathematics, then there there is no greater system and such a proof of the consistency of PM is unprovable.

Not only were the formalists, Russell, Whitehead, and Hilbert losers, they were double losers. Not only did the proof they spent decades searching for not exist, it did so provably, so their entire project was only in vain.

3 Turing's Undecidable

Alan Turing takes a class foundations, where he learns Gödel's Incompleteness theorems. It also contains a description of an unsolved problem we will call "Hilbert's decision problem" or the "Entscheidungsproblem".

Definition 3.1 (Entscheidungsproblem)**.** Define a procedure which takes as input a statement T and determines if T is true or false with respect to a set of axioms.

Hilbert genuinely believed there were no unsolvable problems. Turing was twenty two when he gave a negative answer. First, he had to formalize the notion of computation, and to do so, he invented what we now call the Turing machine. A Turing-machine is a

formalization of computation, much like how logic is a formalization of thought. Next, he described the Church-Turing thesis to convince us that this definition was in fact universal. Following the development of the thesis, he gave a rigorous definition of algorithm.

Definition 3.2 (Decidable Languages ($\mathscr{L}_D(TM)$)**.** Recall that a language $L \subseteq \Sigma^*$ is decidable if there exists a Turing machine M such that

$$x \in L \iff M \text{ accepts } w$$
$$x \notin L \iff M \text{ rejects } w.$$

We can rephrase Hilbert's decision problem as "give a process to decide every language". Following the Church-Turing Thesis, the decidable languages give a characterization of the concept of an "algorithm". A purely mechanical process in which a decision on yes or no is always reached. If every language is decidable, then there exists an algorithm to solve every problem. There do not exist any unsolvable problems. Could every language be decidable? Every problem be solvable? Turing said no, there exist undecidable languages. He did so in two ways.

3.1 A Non-Constructive Proof

Theorem 3. There exists undecidable languages.

Proof. First, notice that the languages decidable by Turing machines are countable. Each language is decidable by many deciders, but each decider decides exactly one language. Let D be the set of all deciders and $\mathscr{L}_D(TM)$ be the decidable languages. We may map each decidable language to exactly one of its deciders to see $|\mathscr{L}_D(TM)| \leq |D|$. By the typewriter principle, we see that $|D|$ is countable. Therefore, so must be the decidable languages.

Next observe that the number of languages is uncountable. If $L \subseteq \Sigma^*$ then $L \in \mathcal{P}(\Sigma^*)$. Since $|\Sigma^*|$ is countable, $|\mathcal{P}(\Sigma^*)|$ is uncountable, by Cantor's theorem. There exists no surjection $\mathscr{L}_D(TM) \to \mathcal{P}(\Sigma^*)$ so there must exist undecidable languages. There exists languages which do not have a Turing machine to decide them. □

By a simple counting argument, we were able to show that most languages, infinitely many more languages are undecidable. This doesn't show the existence of some specific undecidable language, an unsolvable problem. Turing showed that as well.

3.2 Universality

A Turing-complete model of computation is universal. Turing machines are formal objects, but they all have descriptions. Programs can be uniquely determined by their code. Turing originally used Gödel numberings, but we may observe that to each Turing machine, there corresponds a unique string. (Σ^* and $\mathbb{N}$ have the same cardinality, as long as we are injective in our description, it doesn't matter). A Turing machine M can be represented by its description, its encoding $\langle M \rangle$. But Turing machines also compute on strings. We may certainly give a Turing machine as input the description of another.

Definition 3.3 (Universal Turing Machine)**.** There exists a Turing machine U which takes as input a description of a Turing machine $\langle M \rangle$ and a word w and it simulates M on w such that:

$$\begin{aligned} U \text{ accepts } \langle M, w \rangle &\iff M \text{ accepts } w \\ U \text{ rejects } \langle M, w \rangle &\iff M \text{ rejects } w \\ U \text{ loops on } \langle M, w \rangle &\iff M \text{ loops on } w \end{aligned}$$

Such a Turing machine exists by the Church-Turing Thesis. Not only does there exist a universal simulator, but Turing machines may take in descriptions of other Turing machines, simulate them, and have conditional behavior. They may even take on descriptions of themselves.

3.3 An Unsolvable Problem

There exist real, concrete, unsolvable problems. There exists languages which are definable but not decidable.

Definition 3.4. Define $HALT \subseteq \Sigma^*$ as a language where

$$HALT = \{ \langle M, w \rangle \mid M \text{ halts on } w \}$$

$HALT$ is a language of pairs of encodings of Turing machines and possible inputs, where $\langle M, w \rangle \in \text{HALT} \iff$ machine M halts on input w.

We show that $HALT$ is not decidable. This means there is no general algorithm to decide if a Turing machine will halt on an input! A provably unsolvable problem.

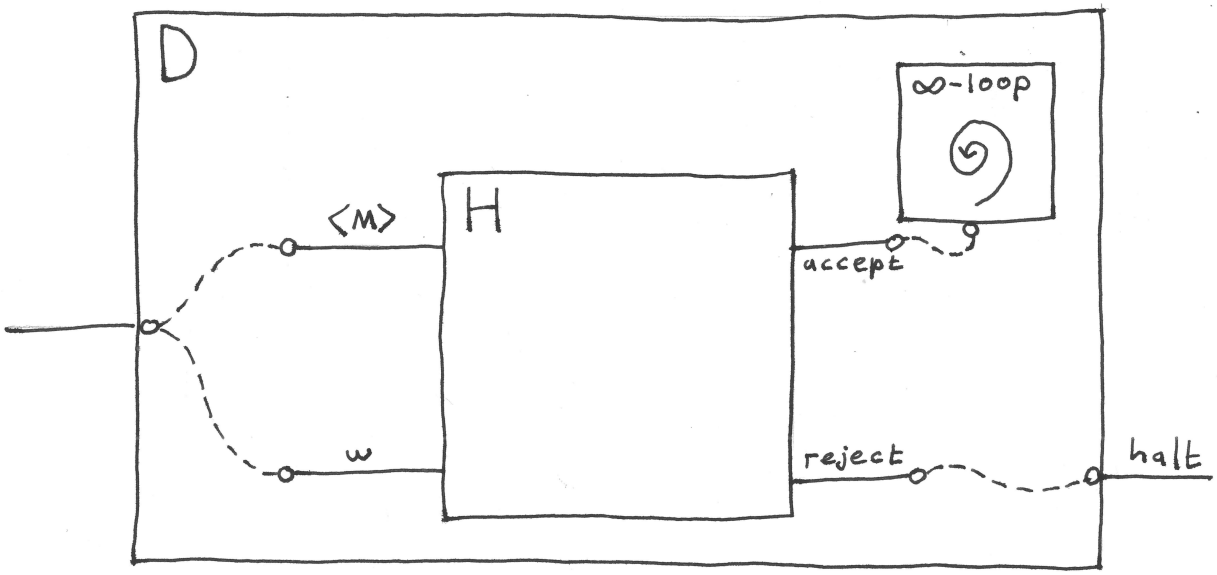
Theorem 4 (The Halting Problem)**.** The language $HALT$ is undecidable.

Proof. Assume to the contrary that $HALT$ is decidable. Then there exists a Turing machine $H(\langle M \rangle, w)$ on input $\langle M \rangle$ and w which always correctly says accepts or rejects if M halts on w. Notice that since H is a decider, on all inputs, it always accepts or rejects and never loops. We build a Turing machine D around H in a black-box way. It uses H as a subroutine. D takes in one argument and passes it to both arguments of H. Then if H returns true, D infinitely loops. If H returns false, then D simply returns and halt.

In pseudocode, D is doing the following with H:

```
def D(M):
    if H(M,M):
        while True:
            continue
    else:
        return
```

Certainly since H exists, then so does D. What is D on input $\langle D \rangle$? $D(\langle D \rangle)$? There is no problem with asking this question. We may run the code of a machine on the machine itself with no problem. Compilers can compile themselves. We have two cases, whether or not D loops or halts on this input.



- $D(\langle D \rangle)$ halts $\iff$ $H(\langle D \rangle, \langle D \rangle)$ rejects $\iff$ $D(\langle D \rangle)$ loops
- $D(\langle D \rangle)$ loops $\iff$ $H(\langle D \rangle, \langle D \rangle)$ accepted $\iff$ $D(\langle D \rangle)$ halts.

A contradiction. No decider for H can exist and we see it is undecidable.

□

This is a proof by diagonalization. You have some negated self-reference. Here a machine is being run on its own code, and does the opposite of what its decidable subroutine says it would do. If you were to create a table with one axis machines, and the other axis their descriptions, then the machine D and description $\langle D \rangle$ would exist somewhere along the diagonal.

4 Conclusion

We have given three proofs in three different settings. If you have a keen eye, you may note these are really all the same proof. They all have the same structure. They are all diagonalization over different settings. Sets, logical formulas, and decidable languages. These three proof all have the tell-tale common features of a proof by diagonalization. There is some negation, and some self-reference, or diagonal. A set not containing itself, a formula saying something about its own unprovability, or a machine contradicting being run on its own code.

5 Moral of History

Both of these theorems are formal, but they involve devices which correspond to our intuitive human processes. Turing machines are a formalization of computation, so the existence of a problem unsolvable by Turing machines implies the problem is unsolvable by humans as well.

Logic is a formalization of thought and deduction, so the existence of a true but unprovable statement implies that there exists truths which we cannot demonstrate certainty about.

What is the moral of the history here? We should be incredibly thankful that Hilbert's program failed. Had it succeeded, mathematics would have been drained of all its creativity. There would exist perfect automatic theorem provers. All of mathematics, all of the complex and beautiful technical arguments could be reduced to symbolic manipulation. Mathematics is an ancient, and didactic, and even dramatic tradition. You sit in front of a board and a lecturer like humanity has for millennia. Reduction of this art to something as mechanical as a combine harvester, reaping theorems, is controversial, putting it politely. Thus ends a millennia long project. The formalization of thought and computation cannot be completed to a satisfactory level. I am personally thankful that the mechanization of mathematics failed. Otherwise, I would not have this job. Some of Hilbert's program has been salvaged. The consensus is that ZFC forms a safe and conservative foundation for much of the usable parts of mathematics. This is independent of Gödel's theorems, which say that ZFC could never be not both complete or consistent.

A second moral is to not bet against the youth. Cantor was 29 when he first proved the existence of an uncountable set. His strongest critic, Kronecker, was 51. Russell was 29 when he showed his Paradox. Frege was 53. Russell chose to go down the same path, attempting to build a system that Frege could not. Gödel was 24 when he showed his incompleteness theorems. By then, Russell had aged to 59. Turing was 24 when he proved the existence of unsolvable problems. Hilbert was 74.[1] It can become easy to become entrenched in your own ideas for decades. All it may take someone younger to come in with a different perspective.

[1] These may be off by a year or two since I don't want to count months.