Gödel's Theorem

Summary Sheet – 24.118, Spring 2021

1 The Theorem

Let $\mathcal{L}$ be a (rich enough) arithmetical language:

Gödel's Incompleteness Theorem (V1) No Turing Machine can do the following: when given a sentence of $\mathcal{L}$ as input, it outputs "1" if the sentence is true and "0" if the sentence is false.

Gödel's Incompleteness Theorem (V2) No Turing Machine can:

1. run forever, outputting sentences of $\mathcal{L}$;
2. eventually output each true sentence of $\mathcal{L}$; and
3. never output a false sentence of $\mathcal{L}$.

Gödel's Incompleteness Theorem (V3) No axiomatization of $\mathcal{L}$ is both consistent and complete.

2 What the Theorem Teaches Us

- *Mathematically:*

  Arithmetical truth is too complex to be finitely specifiable.

- *Philosophically:*

  Interesting mathematical theories can never be established beyond any possibility of doubt.

3 A Simple Arithmetical Language, L

- An **arithmetical language** is a language to talk about the natural numbers and their basic operations (e.g. addition and multiplication).
- L is an arithmetical language built from the following symbols:*

*With some effort, the exponentiation symbol can be defined using "+" and "×", as Gödel showed. I include it here because it will make proving the theorem much easier.

Arithmetical Symbol	Denotes
0	the number zero
1	the number one
$+$	addition
$\times$	multiplication
$\wedge$	exponentiation

Logical Symbol	Read
$=$	... is identical to ...
$\neg$	it is not the case that ...
&	it is both the case that ... and ...
$\forall$	every number is such that ...
x_n (for $n \in \mathbb{N}$)	it

Auxiliary Symbol	Meaning
(	[left parenthesis]
)	[right parenthesis]

3.1 Abbreviations

An **abbreviation** is a notational shortcut to make it easier for us to keep track of certain strings of symbols on our official list.

- *Numerals:*

Abbreviation	Read	Official Notation
2	two	$(1+1)$
3	three	$((1+1)+1)$
4	four	$(((1+1)+1)+1)$
$\vdots$	$\vdots$	$\vdots$

- *Logical Symbols:*

Abbreviation	Read	Official Notation
$A \vee B$	A or B	$\neg(\neg A \ \& \ \neg B)$
$A \supset B$	if A, then B	$\neg A \vee B$
$\exists x_i$	some number is such that it	$\neg\forall x_i\neg$

- *Arithmetical Symbols:*

Abbreviation	Read	Official Notation
$x_i < x_j$	x_i is smaller than x_j	$\exists x_k((x_j = x_i + x_k) \ \& \ \neg(x_k = 0))$
$x_i \mid x_j$	x_i divides x_j	$\exists x_k(x_k \times x_i = x_j)$
Prime(x_i)	x_i is prime	$(1 < x_i) \ \& \ \forall x_j \forall x_k((x_i = x_j \times x_k) \supset (x_i = x_j \vee x_i = x_k))$

4 A Rich Enough Language

$\mathcal{L}$ counts as "rich enough" if one can prove:

Lemma $\mathcal{L}$ contains a formula (abbreviated "Halt(k)"), which is true if and only if the kth Turing Machine halts on input k.

(As it turns out, even our simple language L satisfies this condition!)

5 A Proof of Gödel's Theorem (V1)

- Assume for *reductio*: M decides the truth of sentences of $\mathcal{L}$.
- By the Lemma, we can use M's program to construct a Turing Machine M^H, which computes the Halting Function.
- Since the Halting Function is not Turing-computable, our assumption must be false.

We've proved Gödel's Thoerem!

6 Proving the Lemma

Let's now show that our simple language L counts as a "rich enough" language.

6.1 The key idea

- The key is to be able to express claims about *sequences* in L.
- We need a formula—abbreviated "Seq(c, n, a, i)"— which is true if and only if c encodes a sequence of length n of which a is the ith member.
- With that in place, proving the lemma is totally straightforward.

6.2 Warm Up: Pairs

Coding System

- To the pair $\langle n, m \rangle$ ($n, m \in \mathbb{N}$) assign the number $2^n \cdot 3^m$.

Implementation in L

- Pair(x_i, x_j, x_k) $\leftrightarrow_{df}$ $x_i = (2^{x_j} \times 3^{x_k})$

6.3 Coding Finite Sequiences

Coding System

Part 1:

- Let c's unique decomposition into primes be

$$p_0^{e_0} \cdot p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots p_k^{e_k}$$

  where $p_i \neq p_j$ whenever $i \neq j$ and $e_i \neq 0$.

- We say that c's *non-trivial exponents* are $e_0, e_1, \ldots, e_k$.

- Each number can be thought of a code for the set of its non-trivial exponents.

[This is only half the job, because sets are unordered.]

Part 2:

- Suppose c's non-trivial exponents code ordered pairs, and that each such pair has a different natural number as its first component.

- Then the first components of the pairs can be used to define an ordering of the pairs' second components.

Example:

- $c = 2^{2^2 \cdot 3^{17}} \cdot 5^{2^1 \cdot 3^7} \cdot 7^{2^3 \cdot 3^{117}}$

- c's non-trivial exponents: $\{2^2 \cdot 3^{17}, 2^1 \cdot 3^7, 2^3 \cdot 3^{117}\}$.

- Such a set is code for: $\{\langle 2, 17\rangle, \langle 1, 7\rangle, \langle 3, 117\rangle\}$.

- The first components induce the following ordering of the second components: $\langle 7, 17, 117\rangle$.

- c codes the finite sequence $\langle 7, 17, 117\rangle$.

Implementation in L

We'll divide the problem into two components:

1. Define "Seq(c, n)" [read: c codes an n-sequence].

$$\begin{aligned} \text{Seq}(c, n) \quad \leftrightarrow_{df} \quad & \forall x_i((1 \leq x_i \;\&\; x_i \leq n) \supset \\ & \exists! x_j(\exists x_k(x_j = 2^{x_i} \times 3^{x_k}) \;\&\; \exists x_k(\text{Prime}(x_k) \,\&\, x_k^{x_j} \mid c \,\&\, \neg(x_k^{x_j+1} \mid c))) \end{aligned}$$

   [Read: For each i $(1 \leq i \leq n)$, c's non-trivial exponents include the code for exactly one pair of the form $\langle i, b\rangle$.]

2. Define "Seq(c, n, a, i)" [read: c encodes an n-sequence of which the ith member is a].

$$\begin{aligned}\text{Seq}(c,n,a,i) \quad \leftrightarrow_{df} \quad & \text{Seq}(c,n) \;\&\; \\ & (1 \leq i \;\&\; i \leq n) \;\&\; \\ & \exists x_j \big(\text{Prime}(x_j) \;\&\; (x_j^{(2^i \times 3^a)} \mid c) \;\&\; \neg (x_j^{(2^i \times 3^a)+1} \mid c)\big)\end{aligned}$$

[Read: Seq(c, n) and $(1 \leq i \;\&\; i \leq n)$ and c's non-trivial exponents include a code for $\langle i, a \rangle$.]

7 Gödel's Theorem (v3)

7.1 Axiomatization

- An **axiom** is a sentence that is taken to require no proof.
- A **rule of inference** is a rule for inferring some sentences from others.
- An **axiomatization** for $\mathcal{L}$ is a (Turing Computable) list of axioms and rules of inference for $\mathcal{L}$.

7.2 Provability, completeness and consistency

For $\mathcal{A}$ an axiomatization of $\mathcal{L}$:

- A sentence S of $\mathcal{L}$ is **provable** in $\mathcal{A}$ if there is a finite sequence of sentences of $\mathcal{L}$ such that:
  - Every member of the sequence is either an axiom of $\mathcal{A}$, or results from previous members of the sequence by applying a rule of inference of $\mathcal{A}$.
  - The last member of the sequence is S.
- $\mathcal{A}$ is **complete** if every true sentence of $\mathcal{L}$ is provable in $\mathcal{A}$.
- $\mathcal{A}$ **consistent** if it is never the case that both a sentence of $\mathcal{L}$ and its negation are provable in $\mathcal{A}$.

7.3 Proving the Theorem

- For *reductio*: $\mathcal{A}$ is a consistent and complete axiomatization of L.
- Since L can talk about finite sequences, it can talk about sentences (i.e. finite sequences of symbols) and proof (which are finite sequences of sentences).
- One can program a Turing Machine M to output all and only the sentences of L that are provable in $\mathcal{A}$.
- If $\mathcal{A}$ is consistent and complete, M outputs all and only the true sentences of L, which contradicts Gödel's Theorem (v2).

8 Hilbert's Program

Mathematical Hypothesis There is an algorithmic method capable of establishing, once and for all, whether a set of axioms is consistent.

Philosophical Hypothesis All it takes for a set of axioms to count as a true description of some mathematical structure is for it to be consistent.

The two together: it is possible to establish whether a set of axioms is true by applying an algorithm that tests for consistency.

8.1 Gödel's Bad News

Gödel undermined Hilbert's mathematical hypothesis by proving:

1. **Arithmetization of Turing Machines** If a problem can be solved by a Turing Machine, it can be proved to be true or false on the basis of the standard axiomatization of arithmetic, PA.

2. **Gödel's Second Theorem** If an axiomatic system is at least as strong as PA, it is unable to prove its own consistency (unless it is inconsistent, in which case it can prove anything, including its own consistency).

The two together: if PA consistent, its consistency cannot be established by an algorithmic method.

8.2 Gödel's Theorem: From Truth to Proof

- *We have seen:* there is a formula of L—abbreviated "Computes(n, i, o)"—that is true if and only if the nth Turing Machine yields output o given input i.

- *Gödel also showed:* "Computes(n, i, o)" is **provable** in PA if and only if the nth Turing Machine yields output o given input i. In symbols:

$$\vdash_{PA} \text{Computes}(n, i, o) \Leftrightarrow \ n\text{th TM yields output } o \text{ given input } i.$$

Gödel's Theorem Gödel's Way

1. There is a formula of L—abbreviated "$\text{P}_{PA}(n)$"—that expresses provability in PA:

$$\vdash_{PA} \text{P}_{PA}(n) \Leftrightarrow \text{ the formula coded by } n \text{ is provable in } PA.$$

2. There is a formula of L—abbreviated "$\mathcal{G}$"—that says that it is not provable in PA:[†]

$$\vdash_{PA} \mathcal{G} \leftrightarrow \neg \text{P}_{PA}(\overline{\mathcal{G}})$$

[†]Here and below, $\overline{\phi}$ is a number that codes the formula ϕ.

3. If PA is consistent,[‡] neither $\mathcal{G}$ nor its negation is provable in PA. In particular:

$$\nvdash_{PA} 0 = 1 \quad \Rightarrow \quad \nvdash_{PA} \mathcal{G}$$

4. Gödel's proof can be formalized in L and proved in PA. In particular:

$$\vdash_{PA} \neg \mathrm{P}_{PA}(\overline{0 = 1}) \supset \neg \mathrm{P}_{PA}(\overline{\mathcal{G}})$$

Gödel's Second Theorem

If an axiomatization of arithmetic at least as strong as PA is consistent, it can't prove its own consistency. In particular:

$$\nvdash_{PA} 0 = 1 \quad \Rightarrow \quad \nvdash_{PA} \neg \mathrm{P}_{PA}(\overline{0 = 1})$$

[‡]The notion of consistency that's required to show that the negation of $\mathcal{G}$ is not provable is ω-consistency, which says that you can't prove each of the following for some F: $F(0), F(1), \ldots, \exists x_0 \neg F(x_0)$. One can, however, show that there are other sentences of L such that neither they nor their negations are provable in PA assuming ordinary consistency: $\nvdash_{PA} 0 = 1$.

Proof:

- $\nvdash_{PA} 0 = 1$ (assumption)
- $\nvdash_{PA} \mathcal{G}$ (by 3 above).
- $\nvdash_{PA} \neg \mathrm{P}_{PA}(\overline{\mathcal{G}})$ (by 2 above).
- $\nvdash_{PA} \neg \mathrm{P}_{PA}(\overline{\overline{0 = 1}})$ (by 4 above).