Complexity, Incompleteness, and Foundation of Mathematics

Lingyuan Ye
June 17, 2021

ye.lingyuan.ac@gmail.com

Abstract. The aim of this paper is to show how complexity lies at the heart of foundation of mathematics, in that it puts various constraints on our ability to derive formal proofs. I will argue for this by presenting how various incompleteness results, viz. the existence of various true but unprovable propositions, are deeply connected to complexity theory.

Keywords: Large Complexity Class, Definitional Complexity, Kolmogorov Complexity, Incompleteness, Foundation.

1 Introduction

Human beings perhaps started to do very simple tasks of counting and calculations from the very beginning of our existence as a species. However, it was probably not until the Greek era that we used truly rigorous method of *mathematical reasoning* to derive mathematical knowledge. From then on, in the following roughly 3000 years, mathematics as a field of study flourished. Enormous progress had been made in number theory, geometry, algebra, analysis and applied mathematics (physical sciences). Starting from the end of 19th century, yet another conceptual revolution began in mathematics: the very activity of mathematical reasoning and mathematical proofs themselves had been formalised, and become a subject of scrutiny of mathematics! The study of structures and properties of proofs in various formal logical systems, e.g. systems of arithmetic and axiomatic set theory, can thus be adequately viewed as the study of *meta-mathematics*. These are the main subjects of modern mathematical logic. Furthermore, with the formalised definition of mathematical language and proofs, for the first time in the history we might be able to provide mathematics with a solid foundation.

This line of thoughts perhaps culminated at the 1920s, when David Hilbert famously proposed his programme for providing a powerful enough yet consistent foundation of classical mathematics. The ultimate goal of this programme is to axiomatise most of, if not all, known mathematical knowledge, together with a proof *within the very system itself* that it is consistent. The hope then was that if this task could be carried out eventually, we would have a solid foundation of mathematics, validating different sorts of mathematical activities, as well as guaranteeing their correctness. We refer the readers to the entry for Hilbert's programme on the Stanford Encyclopaedia of Philosophy [Zach, 2007] for an elementary account of it; for a more philosophical as well as historical analysis of the impact of Hilbert's programme on mathematics, see the very nice book by Frank [Franks, 2009].

However, several years later, Gödel published his seminal paper [Gödel, 1931] where he proved his famous incompleteness results. Gödel's incompleteness theorems are usually formulated in the first-order theory of arithmetic, though it actually applies to a much wider range of theories, including much stronger theory of axiomatic set systems like ZF or ZFC (cf. [Kunen, 2014]). The First Incompleteness Theorem basically says that there are *true* arithmetic sentences which are not *provable*. The Second Incompleteness Theorem informally states that for any sufficiently strong system, if itself is consistent then it cannot prove its own consistency. Of course, these formulations are a bit vague and informal, and we'll get to their formal content soon in the following texts. The wide applicability and technical significancy of the incompleteness results had basically showed that Hilbert's programme could never be full carried out — not because the intellectual ability of human is limited, but because there are fundamental mathematical constraint in the very methodology of mathematical proof itself! See [Wang, 1997] for a very nice discussion on the impact of Gödel's results, which is free of many of the philosophical abuse of Gödel's theorems in the literature. Part of the goal of this paper is to provide a complexity-theoretic understanding of Gödel's result, which is the main topic of Section 2. Section 3 provides a further generalisation of the framework discussed in Section 2, and shows more connections between complexity-theoretic questions and incompleteness phenomena.

After Gödel, a lot more incompleteness results have been shown within various different contexts (cf. [Tarski et al., 1953] [Chaitin, 1990]). Among them, the most conceptually significant progress is closely tied to the development of Kolmogorov complexity and its relation with arithmetic. Such insights can be already partly seen in some of the early works of Chaitin's on algorithmic information theory (cf. [Chaitin, 1987]), and other attempts to connect Kolmogorov complexity with Gödel's incompleteness results (cf. [Chaitin, 1996,Kikuchi, 1997,Raatikainen, 1998]). A summary of related topics can be found in [Chaitin, 1996]. Much more recently, [Kritchman and Raz, 2010] has proved the Second Incompleteness Theorem using the interplay between Kolmogorov complexity and arithmetic, which provides further insights into the deep connection between complexity theory and logic.

The significance of the perspective of Kolmogorov complexity is that, it shows there is an *intrinsic complexity barrier*, which is quite low actually, within any formalised system. Section 4 will show that this complexity barrier accompanied every formal logical system puts a more fundamental restriction on what we can prove by formal methods, and perhaps can shed more light on our ability to obtain mathematical knowledge through reasoning.

The remaining part of the paper will be structured as follows: Section 2 will introduce the basic framework of first-order arithmetic, and show Gödel's incompleteness theorems with the perspective of complexity theory, or more precisely recursion theory. Section 3 generalise the results obtained in the previous section by showing the relation between arithmetic and a hierarchy of large complexity classes. In particular, I will use arithmetic hierarchy to define levels of large complexity classes, and show how to view this relation between incompleteness results in formal logic and complexity theory in a more conceptual way. Section 4 will extend such a tie between arithmetic and complexity theory by showing implications of Kolmogorov complexity for formal logical systems. I will show that the Kolmogorov complexity sets up an essential and inherent constraint for any formal system, that fundamentally restricts its ability to generate proofs. At the end, Section 5 will sum up the formal results obtained in the previous sections, and give a briefly discussion of their implications for the foundation of mathematics. The overall goal of this essay is to provide a deeper understanding of the limit of formal systems in the light of complexity.

2 Recursion Theory and Gödels Incompleteness Theorem

In this section we will briefly review the contents of Gödel's incompleteness theorems, and show how they are deeply related to some complexity-theoretic concepts, mainly in recursion theory. I will not provide a full introduction to first-order arithmetic, since that will take up too much space and exceed the goal of the current essay, but I will try to explain in a semi-formal manner for most of the notions I will be discussing.

2.1 Peano Arithmetic and First Incompleteness Theorem

As already mentioned, Gödel's incompleteness theorems is usually formulated in the formal first-order language of arithmetic; let us denote it as $\mathcal{L}$. $\mathcal{L}$ consists of the following syntactical ingredients:

(1) variables: $x, y, z, \cdots$;
(2) function symbols: $0, \mathrm{S}, +, \times$ or arity 0, 1, 2 and 2, respectively; intuitively, they represent the constant 0, successor function, addition and multiplication;
(3) logical constants: $\neg, \rightarrow, \forall, =$;
(4) auxiliary symbols: (,).

In the above definition, symbols contained in (2) are specific to our arithmetic language $\mathcal{L}$; others are present in any first-order theory. Using these syntactical elements, we can build up terms and formulas within our language $\mathcal{L}$. I will not present a formal definition for what are well-defined terms and formulas in $\mathcal{L}$, but illustrate them by showing you some examples.

For instance, for any natural number n, we can build the following term

$$\overline{n} := \underbrace{\mathrm{S} \cdots \mathrm{S}}_{n \text{ times}} 0$$

Notice that the symbol $\overline{n}$ is actually an abbreviation for the actual term $\mathrm{S} \cdots \mathrm{S}0$. The future definitions of this kind should always be understood in this way. $\overline{n}$ is called the *numeral* of the number n in $\mathcal{L}$. If we are presented with terms t, s in $\mathcal{L}$, we could further build $\mathrm{S}t$, $t + s$ and $t \times s$ as new terms.

Given any two terms $t, s, t = s$ is a basic formula. We can build up more complex formulas by combining them with logical connectives. For any formulas φ, ψ, we can construct new formulas out of them, including $\neg\varphi, \forall x\varphi, \varphi \to \psi$, etc. We consider other logical connectives as abbreviations. For example, $\varphi \wedge \psi$ stands for $\neg(\varphi \to \neg\psi)$; $\varphi \vee \psi$ stands for $\neg\varphi \to \psi$; and for any variable x, $\exists x\varphi$ stands for $\neg\forall x\neg\varphi$. Such a choice is not essential, but it will simplifies our future formulation of formal proofs.

A variable appears *freely* in a formula φ if it is not contained in the scope of a corresponding quantification. Consider the formula

$$\forall x(x + y = y \to \exists z(z = x))$$

The variable x and z are bounded by the corresponding universal and existential quantifier; the only variable that appears free is y. Given any formula φ, we use the notation $\varphi(x_1, \cdots, x_k)$ to denote that the free variables appearing in φ are contained in $x_1, \cdots, x_k$. A formula is called a *sentence* if it contains no free variables. Similarly, we use the notation $t(x_1, \cdots, x_k)$ to denote the fact that the variables appearing in t are contained in $x_1, \cdots, x_k$. A term which contains no free variables will be called *closed.*

Using the vocabulary of $\mathcal{L}$, we can indeed express most of definitions about numbers. For example, we can define the linear relation $\leq$ as follows:

$$x \leq y := \exists z(x + z = y)$$

Of course, $x < y, x > y, x \geq y$ can all be easily defined using $x \leq y$. The devides relation can be defined as well,

$$x \mid y := \exists z(x \times z = y)$$

It is interesting to notice the structural similarity between $x \leq y$ and $x \mid y$. Using the above defined notion, we can express what it means to be a prime number:

$$\mathsf{prime}(x) := \overline{2} \leq x \land \forall y(y \mid x \to y = \overline{1} \lor y = x)$$

The above examples should give you a hint of the expressive power of the first-order arithmetic language $\mathcal{L}$.

Up till now, everything is about pure syntax and the structure of terms and formulas. To further define what is a formal proof in our language $\mathcal{L}$, we need to specify two additional things: axioms and rules of deduction. Intuitively, a proof of ψ is then a finite sequence of formulas such that each formula appearing in the sentence is either an axiom or can be derived from previous formulas according to some rule of inference, and the sequence ends at ψ.

The most common axiomatisation of a quite strong arithmetic system is Peano Arithmetic, and let us denote it as PA. The axioms of PA are separated into two groups. First we have *logical axioms*:

L1. $\varphi \to (\psi \to \varphi)$;
L2. $(\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))$;
L3. $(\neg\psi \to \neg\varphi) \to ((\neg\psi \to \varphi) \to \psi)$;
L4. $\forall x\varphi(x) \to \varphi(t)$, for any closed term t;
L5. $\forall x(\psi \to \varphi(x)) \to (\psi \to \forall x\varphi(x))$;
L6. $\forall x x = x$;
L7. $\forall x \forall y(x = y \to (\varphi(x) \leftrightarrow \varphi(y)))$

Actually, the above presented ones are not single axioms, but *axiom schemes*. For any instance of formulas φ, ψ, χ and closed term t, there is a corresponding axiom of the above form. Hence, they represent infinitely many axioms. These logical axioms are actually present in any first-order theories, since they are supposed to be valid by purely logical means. Besides these, we also have axioms specific to PA:

A1. $\forall x(\neg Sx = 0)$;
A2. $\forall x \forall y(Sx = Sy \to x = y)$;
A3. $\forall x(x + 0 = x)$;
A4. $\forall x \forall y(x + Sy = S(x + y))$;
A5. $\forall x(x \times 0 = 0)$;
A6. $\forall x \forall y(x \times Sy = (x \times y) + x)$;
A7. $\varphi(0) \land \forall x(\varphi(x) \to \varphi(Sx)) \to \forall x\varphi(x)$

Basically, A1 - A2, A3 - A4 and A5 - A6 axiomatise the recursive definition of successor, addition and multiplication, respectively. A7 is again an axiom scheme, representing mathematical induction. Since we have induction for any formula $\varphi(x)$, within PA we can develop a great amount of arithmetic.

The only rules for deduction in first-order logic are Modens Ponens (MP) and universal generalisation (R$\forall$):[1]

MP: from φ and $\varphi \rightarrow \psi$ we can derive ψ;
R$\forall$: from $\varphi(x)$ we can infer $\forall x \varphi(x)$.

As we've already mentioned, a proof of ψ in PA then is a finite sequence of formulas, where you start from axioms of PA, either logical or non-logical, and derive ψ in a finite number of steps according to the above rules of inference. Whenever a formula ψ is derivable from PA, we write PA $\vdash \psi$, and we say ψ is a *theorem* of PA.

Such a formal presentation of proof system allows you to derive all the familiar facts about arithmetic, e.g. the commutativity and associativity of addition and multiplication, every non-zero element has a predecessor, the reflexivity and transitivity of the defined relation $\geq$, etc., etc.. Basically, everything you can prove in basic arithmetic from elementary means by using mathematical induction can be formalised in our proof system PA. As previously mentioned, the benefits of such a formalisation is that now we have a rigorous notion of what counts as a proof, and we can then study their properties and structures by mathematical means.

Our formalisation of PA is intended to reason about natural numbers. Intuitively, for any sentence ψ in our language $\mathcal{L}$, it expresses some property about the set of natural numbers, and it can either be *true* or *false* in $\mathbb{N}$. If ψ is true in $\mathbb{N}$, we write $\mathbb{N} \models \psi$. For example, the sentence $\forall x(x = 0)$ is obviously false, while $\forall x(0 + x = x)$ is obviously true. Such a definition can again be made precise, but we will not carry it out in its full length here in this paper, and we think we can safely assume most of the readers will have an intuitive grasp of this notion. We refer the readers to any standard textbook of first-order logic for a more rigorous account.

For our above formalisation of proofs to ever be considered as meaningful, one property we would like it to have is that whatever we can prove must be true. This is the soundness lemma of first-order arithmetic:

Lemma 1 (Soundness). *For any sentence ψ,* PA $\vdash \psi$ *implies* $\mathbb{N} \models \psi$.

Proof (Sketch). It is not hard to convince oneself that every axiom of PA is true, and the two derivation rules preserves truth. □

The more interesting question to ask is whether what is true in $\mathbb{N}$ can be provable or not. Gödel's First Incompleteness Theorem shows the answer is negative:

Theorem 1 (First Incompleteness Theorem). *There exists a sentence γ in $\mathcal{L}$ such that* $\mathbb{N} \models \gamma$ *but* PA $\nvdash \gamma$.

[1] This is due to the fact that we have assumed other logical connectives are certain abbreviations of $\neg, \rightarrow, \forall$. If you treat other connectives as primitive symbols, you will have to add more deduction rules.

The original proof presented in [Gödel, 1931] of the First Incompleteness Theorem is very interesting, because the way Gödel proved this is to formalise a paradox, viz. the liar paradox, within our formal first-order system of arithmetic. However, in the following texts we will not be following Gödel, but present a complexity-theoretic perspective on Gödel's result, which for the first time hints at a very close tie between logic and complexity theory.

2.2 Recursion Theory and Incompleteness

In a word, a recursion theoretic understanding of the First Incompleteness Theorem is that the complexity of the set of provable sentences is greatly lower than the complexity of the set of sentences that are true. But to make this precise, we first need to provide some recursion theoretic definitions.

Definition 1. *We say a subset $S \subset \mathbb{N}$ is recursive (computable, decidable) if there is a Turing machine (TM hereafter) that decides S in a finite amount of time, i.e. there is a total Turing machine M such that for any input $n \in \mathbb{N}$, M accepts if it is an element of S and rejects if it is not.*

From Definition 1 it is easy to see that every finite subset is obviously recursive. For any finite subset S there is a TM that simply compare the input with every element in S and output the results of comparison. Also, if S is recursive then so does its complement $\mathbb{N}\backslash S$, because we can simply interchange the output of the TM that decides S. However, we already know from basic computability theory that not every infinite subset is recursive. The set of halting TM on a specific input — with respect to a particular coding of TM by numbers — is a subset of $\mathbb{N}$ and it is not recursive. Also, notice that since we have a family of computable pairing functions,

$$\langle - \rangle^k : \mathbb{N}^k \to \mathbb{N}, \ \forall k \in \mathbb{N}_+$$

Definition 1 actually further tells us what are the recursive subsets of $\mathbb{N}^k$ for all k. A subset $S \subset \mathbb{N}^k$ is recursive iff its image under the pairing function $\langle S \rangle^k$ is recursive. If we identify k-ary relations natural numbers as subset of $\mathbb{N}^k$, this provides what it means for a k-ary relation to be recursive.

The original definition of recursive sets, or more generally recursive functions, is actually not in this form. Partial recursive functions are defined to be the class of functions that you can obtain by applying finitely many times of composition, primitive recursion and the so-called μ-operation to a set of basic class of functions that are intuitive computable, including the zero function, the successor function and projections. Recursive functions in this sense are defined to capture the intuitive notion of an algorithm, which at least dated back to Leibniz (cf. [Peckhaus, 1997]).

The fact that the recursive functions defined in this sense coincide with Turing computable functions is actually quite an important theorem in the early development of computability theory, which shows that computational models of very different kind would actually result in the same class of functions (cf. [Rogers, 1987]).

Another class of subsets that is important to our goal here is the *recursively enumerable* ones:

Definition 2. *We say a subset $S \subseteq \mathbb{N}$ is recursively enumerable if there is a TM M such that given any input $n \in \mathbb{N}$, M accepts if it is in S, and may reject or run forever if it is not.*

From Definition 2 it is obvious to see that every recursive subset is recursively enumerable (r.e. hereafter). However, not every r.e. subset is recursive. Hence, the class of recursive subsets is properly contained in the class of r.e. subsets. For one concrete example of r.e. but not recursive subsets see the proof of Proposition 2; basically, the set of halting TM is not recursive but r.e.. Again, Definition 2 gives a more general notion of r.e. subsets of $\mathbb{N}^k$ for any finite k.

There are many equivalent formulations of r.e. subsets; every one of them shows us some particular features of this complexity class. The following lemma presents some of the equivalent characterisations:

Lemma 2. *A subset S is r.e. iff any one of the following holds,*

(1). There is a TM M whose set of inputs that halt is precisely S;

(2). There is a TM M that lists all the members of S, i.e. it outputs $s_1, s_2, \cdots$ as a list where each s_i is an element of S, and each element of S appears as some s_i in the list. If S is infinite, then M runs forever;

(3). There is a recursive binary relation R such that $n \in S$ iff there is an $m \in \mathbb{N}$ that $R(m, n)$ holds.

Proof. (1) is easy. If a subset S satisfies (1) with a TM M, then we can let a universal TM U simulates M. Whenever M halts we let U output 1; if M does not halt then U would not halt. Hence, S is now r.e. by Definition 2. On the other hand, if we have a r.e. subset S computed by a TM M, we again find a universal TM U simulating M. If M halts or loop forever we do nothing; if M rejects then we add a trivial loop so that U again runs forever. The set of inputs that halts for U would then precisely be S, hence S satisfies (1).

If a subset S satisfies (2) with a TM M, we can obviously construct another TM M', such that given any input n, we run M to generate the list of elements of S and M' finds whether n is in that list. If $n \in S$, then it must appear at somewhere in the list and this implies M' halts at a finite number of steps. If $n \notin S$, then M' would run forever. By (1), this means S is r.e. For the remaining part, starting from a r.e. subset S which is equal to the halting domain of a TM M, we can use a technique called dovetail in algorithm designs to construct another TM M' that enumerates all the elements in S. The computation of M' works as follows:

1. Simulates the first step of $M(0)$;
2. Simulates the first step of $M(1)$;
3. Simulates the second step of $M(0)$;
4. Simulates the first step of $M(2)$;
5. Simulates the second step of $M(1)$;
6. Simulates the third step of $M(0)$;
7. Simulates the first step of $M(3)$;
8. …

Now in the process of simulation, whenever $M(n)$ halts at some step, we add n to the output of M'. The resulting TM M' would then generate all the halting input of M, and thus S now satisfies (2).

Now for (3), suppose S is recursively enumerable equivalent to the halting domain of M_S. We can simply define a binary relation R to be the following,

$$R := \{\ (m, n) \mid M(n) \text{ halts within } m \text{ steps }\}$$

Now obviously, by definition we have

$$n \in S \Leftrightarrow \boldsymbol{\exists} m R(m, n)$$

Here $\boldsymbol{\exists}$ represents the meta-mathematical existential quantifier.[2] Also, notice that R is recursive, becaues we can construct a TM M_R that when given the input $\langle m, n\rangle$ it simulates M_S for m steps; if M_S halts then M_R outputs 1; and if not, M_R outputs 0. M_R is then a total TM that computes R, thus R is recursive. On the other hand, if we have such a recursive relation R computed by M_R such that $n \in S$ iff there exists m that $R(m, n)$ holds, we can simply design a TM M_S with halting domain S: Given an input n, let M_S run M_R with input $\langle 1, n\rangle, \langle 2, n\rangle, \cdots$ successively. If $n \in S$, then by definition for some m we have $M_R(\langle m, n\rangle)$ halts, and thus M_S halts; if $n \notin S$ then M_S runs forever. Hence, by (1) again S is r.e. □

The name of recursive enumerable subsets originally comes from the property (2) above. What Definition 2 presents is usually called *semi-decidable*. However, Lemma 2 shows that these two notions are at least extensionally equivalent, thus can be used interchangeably. The third equivalent formulation of r.e. subsets would be very useful in our following development. The following shows that recursive sets are exactly those that are r.e. and whose complement is also r.e.:

Lemma 3. *If a subset $S \subseteq \mathbb{N}$ is r.e., and the complement $\mathbb{N} \backslash S$ is also r.e., then S is recursive.*

[2] The quantifier $\exists$ is the one within our formal language, and $\boldsymbol{\exists}$ can be roughly identified as the one we use in ordinary English. Distinguish the two is very important for many technical and philosophical reasons.

Proof. Suppose S is enumerated by a TM M_1 and $\mathbb{N}\backslash S$ is enumerated by another TM M_2. Then we can construct a TM M, such that when given an input n it simulates alternatively the computation of M_1 and M_2, and find which one of them outputs n. Since n must appear somewhere in the output of M_1 or M_2, M must halt at some finite number of time. If n appears in M_1 then we let M accept; otherwise, we let M reject. This way, M decides S, thus S is recursive. □

We are now in a position to show how Gödel's Incompleteness Theorem is related to recursion theory. One important technical development by Gödel is that he realised we can represent and develop our syntax for first-order arithmetic within natural numbers itself![3] The first step towards this is to find a *coding* for our syntax.

Definition 3. *A pre-coding for syntax in $\mathcal{L}$ is an injective function*

$$\alpha : \mathcal{L}^* \to \mathbb{N}$$

that assigns each expression e (a finite string of symbols in $\mathcal{L}$) to a number $\alpha(e)$. A pre-coding for $\mathcal{L}$ is a coding if it is also effective.

The notion of effectiveness is similar to that of computability. A pre-coding α is effective iff there is a modified version of TM, whose input tape can read symbols from $\mathcal{L}$ and whose output tape can write symbols representing numbers.

Fix a coding function α, we can then talk about the complexity of subsets of syntactical objects. Given a subset S of $\mathcal{L}^*$, we say it is recursive (resp. r.e.) if the set $\alpha(S)$ is recursive (resp. r.e.). Basically, a coding allows you to transfer the intuitive notion of computation in the domain of syntax to the well-studied notion of computation in number theory. For example, all the basic class of syntactical objects, including formulas, terms, sentences, closed terms, etc., are all recursive (computable), as they should be intuitively. Moreover, the set of all axioms of PA is also recursive,

Lemma 4. *Let* $\mathrm{Ax}(\mathsf{PA})$ *be the set of all axioms of* PA*, then the set* $\mathrm{Ax}(\mathsf{PA})$ *is recursive.*

Proof. Since we've already presented axioms of PA as a finite list of axiom schemes, it is obvious that we can write an algorithm that decides whether a given expression has one of the forms presented in the axiom schemes. Hence, by the computable coding function α the set $\mathrm{Ax}(\mathsf{PA})$ would be recursive. □

A more important insight about the complexity of proofs our formal system PA is able to generate is the following:

[3] This idea of coding a class of objects using numbers had played a crucial role in the early development of computability theory. See [Kleene, 1952] for more on this.

Proposition 1. *Let* Th(PA) *be the set of all theorems of* PA*,*

$$\mathrm{Th}(\mathsf{PA}) := \{\, \psi \mid \mathsf{PA} \vdash \psi \,\}$$

then the set Th(PA) *is r.e.*

Proof. Here we use the third characterisation of r.e. subsets in Lemma 2. Notice that we can define a recursive relation $\mathsf{Prf}(m, n)$, such that for any sentence ψ that $\alpha(\psi) = n$, we have

$$\mathsf{Prf}(m, n) \text{ holds} \Leftrightarrow m \text{ is the code of a proof of } \psi$$

Notice that, as we've mentioned a proof is nothing more than a sequence of formulas, thus can be coded by the pairing function $\langle - \rangle$ and the coding α. The relation $\mathsf{Prf}(m, n)$ is indeed recursive because to test whether m codes a proof of ψ (whose code is n), we only need to check the following things: (1) whether m is a sequence of numbers; (2) whether each number in the sequence codes a formula; (3) whether the sequence of formulas constitutes a proof (whether each formula is an axiom or can be derived from previous formulas in the sequence by according to the two derivation rules); (4) whether the last formula appearing in the sequence is ψ or not. All the checking processes appearing in the steps (1)-(4) are recursive, thus $\mathsf{Prf}(m, n)$ is a recursive relation. Now by definition, ψ is provable iff there is a proof in PA that proves ψ, hence we have

$$n \in \mathrm{Th}(\mathsf{PA}) \Leftrightarrow \exists m \mathsf{Prf}(m, n)$$

By Lemma 2, Th(PA) is r.e. □

In a word, the above proof of Th(PA) is r.e. essentially uses the fact that checking whether m codes a proof of ψ is indeed recursive, which then depends on the fact that our set of axioms is recursive and our rules of derivation are finite. Using these two facts, you can also directly write down an algorithm that enumerates all theorems of PA, by enumerating all sequences of formulas according to their length, and for each case check whether that sequence is a proof or not. Proposition 1 gives us a first hint of the complexity of our formal proof system PA.

To show Gödel's First Incompleteness Theorem, we need to further consider the other set of true arithmetic sentences. We denote it as $\mathbb{N}(\mathcal{L})$,

$$\mathbb{N}(\mathcal{L}) := \{\, \psi \mid \mathbb{N} \models \psi \,\}$$

According to the soundness lemma (Lemma 1), Th(PA) is contained in $\mathbb{N}(\mathcal{L})$. Gödel's Incompleteness Theorem shows that Th(PA) is properly contained in $\mathbb{N}(\mathcal{L})$, and in [Gödel, 1931] he explicitly constructed such a sentence γ that is true in $\mathbb{N}$ but not provable. Here, we want to show $\mathrm{Th}(\mathsf{PA}) \subsetneq \mathbb{N}(\mathcal{L})$ by showing the complexity of $\mathbb{N}(\mathcal{L})$ is higher than the complexity of $\mathrm{Th}(\mathsf{PA})$, i.e. $\mathbb{N}(\mathcal{L})$ is not r.e.

Intuitively we would also not expect the set $\mathbb{N}(\mathcal{L})$ to be r.e., because almost all problems can be essentially reduced to arithmetic ones, under a suitable coding. If $\mathbb{N}(\mathcal{L})$ is indeed r.e., we would then be able to decide many previous undecidable questions. But to be more precise, we first need to show a further relation between complexity and our arithmetic theory. The first step is to realise that some subsets of natural numbers are definable by arithmetic formulas in the following sense:

Definition 4. *A subset $S \subseteq \mathbb{N}$ is definably by an arithmetic formula $\varphi(x)$ if the following holds for any $n \in \mathbb{N}$,*

$$n \in S \Leftrightarrow \mathbb{N} \models \varphi(\overline{n})$$

Many usual subsets we encounter in elementary arithmetic are definable. The total set $\mathbb{N}$ can be defined by the formula $x = x$, and the empty set can be defined by $\mathrm{S}x = 0$. For a slight more non-trivial example, previuosly we have shown that the set of prime numbers are also definable. In fact, we can show that every recursive (computable) subset is definable:

Lemma 5. *Every recursive subset $R \subseteq \mathbb{N}$ is definable in $\mathcal{L}$. Explicitly, if R is recursive, then there exists a formula $\varphi_R(x)$ such that for any $n \in \mathbb{N}$, we have*

$$R(n) \text{ holds } \Leftrightarrow \mathbb{N} \models \varphi_R(\overline{n})$$

Proof (Sketch). Here we use the original formulation of recursive functions. As we've mentioned, a recursive function is one that can be obtained by a finite number of applications of composition, primitive recursion and μ-operation to the class of basic functions, including the zero function, projections and successor function. It is obvious that all basic functions are definable. Without two much work you can also show the remaining three operations are also definable. Hence, all recursive functions are definable, as well as recursive subsets. □

An alternative proof of the above fact is to formalise TM in arithmetic. We can use the logical structures of formulas to mimic the computation of TM, and show every computation that can be carried out by TM has a counterpart in the language of arithmetic.

Notice that, again due to the existence of the pairing function, it is easy to see that for any recursive subsets $R \subseteq \mathbb{N}^k$, we can find a formula with k free variables $\varphi_R(x_1, \cdots, x_k)$, such that

$$R(n_1, \cdots, n_k) \text{ holds} \Leftrightarrow \mathbb{N} \models \varphi_R(\overline{n_1}, \cdots, \overline{n_k})$$

A simple corollary from the above fact is that recursive enumerable sets are also definable in our language:

Corollary 1. *For every recursive enumerable sets $E \subseteq \mathbb{N}$ there exists a formula $\varphi_E(x)$ such that for any $n \in \mathbb{N}$,*

$$E(n) \textit{ holds} \Leftrightarrow \mathbb{N} \models \varphi_E(\overline{n})$$

Proof. By Lemma 2, if E is r.e. then there exists a recursive relation $R \subseteq \mathbb{N}^2$ that

$$E(n) \Leftrightarrow \exists m R(m,n)$$

Then the formula $\exists y \varphi_R(y,x)$ defines E,

$$E(n) \text{ holds} \Leftrightarrow \mathbb{N} \models \exists y \varphi_R(y,\overline{n})$$

This means E is also definable. □

Actually, in the next section we will show that recursive (resp. r.e.) subsets are definable by a special class of formulas called Δ_1 (resp. Σ_1) formulas, which shows their relatively low complexity in a hierarchy of large complexity classes. For now, Corollary 1 alone is able to show $\mathbb{N}(\mathcal{L})$ cannot be r.e.:

Proposition 2. *The set $\mathbb{N}(\mathcal{L})$ is not r.e.*

Proof. Recall that abstractly, a Turing machine is tuple (Γ, Q, δ) such that Γ is the alphabet, Q is a finite set of states of the TM, and δ is the transition function. Since all the three sets are finite, we can easily code the TM by numbers

$$\beta : \text{TM} \to \mathbb{N}$$

Given a code $\beta(M)$ of a TM M, we can then effectively compute the corresponding tuple $(\Gamma_M, Q_M, \delta_M)$. Then we can construct a recursive relation $\mathsf{Halt}_0(m,n)$ such that it holds iff the TM coded by n halts within $m+1$ steps when given the input 0. This is recursive because the β codes allows us to decide the transition function and the states of the corresponding TM, and we can then apply $m+1$ times of the transition function with input 0 and check whether the resulting state is the halting state or not. This shows that the unary predicate $\mathsf{Halt}_0(n)$ (with an abuse of notation) expressing that n is the code of a TM that halts when given the input 0 can be defined as follows,

$$\mathsf{Halt}_0(n) \Leftrightarrow \exists m \mathsf{Halt}_0(m,n)$$

and Halt_0 would be recursively enumerable. By Corollary 1, this implies that there exists a formula $\varphi_H(x)$ in $\mathcal{L}$ that defines Halt_0,

$$\mathsf{Halt}_0(n) \text{ holds} \Leftrightarrow \mathbb{N} \models \varphi_H(\overline{n})$$

Now let us suppose $\mathbb{N}(\mathcal{L})$ is r.e.. Since for any sentence ψ, ψ is either true or false in $\mathbb{N}$, hence either ψ or $\neg\psi$ is in $\mathbb{N}(\mathcal{L})$. If we have a TM that enumerates $\mathbb{N}(\mathcal{L})$, we can construct another TM that simply finds $\varphi_H(\overline{n})$ or $\neg\varphi_H(\overline{n})$ in this list, and it must halt as some finite step. We then have obtained a TM that decides Halt_0 which shows its computable, contradictory. Thus, $\mathbb{N}(\mathcal{L})$ is not r.e.. □

From the above proof it should be clear that there are many more such similar proofs that shows $\mathbb{N}(\mathcal{L})$ cannot be r.e.. The general idea behind this is that the language of first-order arithmetic is so expressive such that almost all problems we encounter in ordinary mathematics can be reduced to arithmetic claims — this also explains why number theory is the crown of mathematics. If the complexity of $\mathbb{N}(\mathcal{L})$ is relatively low, then we can use it to decide many questions that are proven to be harder. Hence, the complexity of $\mathbb{N}(\mathcal{L})$ must be greater than most of the problems we face in mathematics — a point will be further investigated in the next section.

Now, we have a pure complexity-theoretic proof of Gödel's First Incompleteness Theorem, and actually a slightly stronger version of that:

Theorem 2 (Extended First Incompleteness Theorem). *There exists infinitely many true arithmetic sentences that are not provable.*

Proof. By Lemma 1, $\text{Th}(\mathsf{PA}) \subseteq \mathbb{N}(\mathcal{L})$. However, Lemma 1 shows $\text{Th}(\mathsf{PA})$ is r.e. but Lemma 2 shows $\mathbb{N}(\mathcal{L})$ is not. Hence, $\text{Th}(\mathsf{PA})$ must be a proper subset of $\mathbb{N}(\mathcal{L})$. The difference of $\text{Th}(\mathsf{PA})$ and $\mathbb{N}(\mathcal{L})$ must also be infinite, otherwise $\mathbb{N}(\mathcal{L})$ would also be r.e.. Hence, there are infinitely many true but unprovable sentences. □

3 Arithmetic Hierarchy & Large Complexity Classes

In the previous section we have shown how Gödel's First Incompleteness Theorem can be understood from recursion theoretic point of view. The important insight is that the set of theorems $\text{Th}(\mathsf{PA})$ of PA is r.e., but the set of true arithmetic sentences $\mathbb{N}(\mathcal{L})$ is not. There, the fact that recursive and r.e. subsets are definable in arithmetic is used essentially in the whole proof. In this section, we will further go down this road and investigate the relationship between complexity and arithmetic theory in greater details. We will define a new hierarchy of large complexity classes through their definability by arithmetic formulas. This complexity would be called *definitional complexity*. We will use this new classification of complexity classes to show some interesting results related to computability theory.

Since we are going to use definability as our classification of complexity classes, we first need to study the complexity of formulas themselves. The complexity of arithmetic formulas are measured by their place in the *arithmetic hierarchy*. We use the following abbreviations of quantifiers:

$$\forall y < x(\psi) := \forall y(y < x \to \psi),\ \exists y < x(\psi) := \exists y(y < x \land \psi)$$

Such quantifications are called *bounded.* Now if a formula φ only contains bounded quantifiers, it is assigned both the classifications Σ_0 and Π_0. We close the classification class under provable equivalence relations. If a formula ψ is provably equivalent to a formula φ which only contains bounded quantification,

$$\mathsf{PA} \vdash \psi \leftrightarrow \varphi$$

we also say ψ is Σ_0 and Π_0. We then recursively define the remaining arithmetic hierarchy as follows:

- If φ is equivalent to a formula of the form $\exists y_1 \cdots \exists y_k \psi$, where ψ is Π_n, then φ is said to be Σ_{n+1};
- If φ is equivalent to a formula of the form $\forall y_1 \cdots \forall y_k \psi$, where ψ is Σ_n, then φ is said to be Π_{n+1};

For a formula φ logically equivalent to a formula of the form

$$Q_1 x_1 \cdots Q_k x_k \psi$$

where Q_i ranges over $\exists, \forall$ and ψ only contains bounded quantification, if Q_1 is $\exists$ (resp. $\forall$) and there are n alternations of existential and universal quantification in the string $Q_1 x_1 \cdots Q_k x_k$, then φ is Σ_n (resp. Π_n). If a formula φ is both equivalent to a Σ_n and Π_n formula, it is then said to be Δ_n. For example, a formula only contains bounded quantification is Δ_0. Notice that if φ is a Σ_n formula, then $\neg\varphi$ is a Π_n formula. This is because when we add a negation in front of a sequence of quantification, pushing the quantification into the back of the formula will change universal quantifier into existensial ones, and vice versa. Since we can simply add dummy quantifications in front of a formula, if a formula φ is Σ_n or Π_n, it also belongs to higher classes in the hierarchy.

An important theorem in classical arithmetic theory is that every formula appears somewhere in the hierarchy:

Lemma 6 (Prenex Normal Form). *Every arithmetic formula φ is logically equivalent to the a formula of the following form,*

$$Q_1 x_1 Q_2 x_2 \cdots Q_k x_k \psi$$

where Q_i ranges over $\exists, \forall$, and ψ is a Δ_0 formula.

Proof. We can prove by induction on the form of our formula φ:

- For a negated formula $\neg\forall x \psi$, it is equivalent to $\exists x \neg\psi$; similarly, $\neg\exists x \psi$ is equivalent to $\forall x \neg\psi$;

– For a conjunction of the form $\exists x\varphi\wedge\psi$, it is then equivalent to $\exists x(\varphi\wedge\psi)$, provided x is not a free variable in ψ (this can always be made true due to a renaming of free variables). Similarly, $\forall x\varphi \wedge \psi$ is also equivalent to $\forall x(\varphi \wedge \psi)$, which also realies on the existence of a constant 0 in PA.

Now since $\neg$ and $\wedge$ are a complete set of Boolean connectives, i.e. every Boolean connective can be expressed by $\neg$ and $\wedge$, it follows that we can recursively do the above procedure to push all the unbounded quantification appearing in the formula to the front. Such a procedure must end for a finite number of steps because every formula is of finite length. Hence, we have actually proved that every formula is equivalent to a form

$$Q_1x_1Q_2x_2\cdots Q_kx_k\psi$$

The remaining untouched quantifiers in ψ must all be bounded, hence ψ is Δ_0. □

It is interesting to point out that the existence of the prenex normal form for every arithmetical formula heavily relies on classical logic. If we work in an intuitionistic setting, the above lemma no longer holds. Lemma 6 has shown us that every arithmetic formula appears somewhere in the arithmetic hierarchy.

Such a classification of complexity of formulas already gives us better understanding of our arithmetic theory. For example, in the previous section we have shown that there are infinitely many sentences that are true but not provable; but with the definition of arithmetic hierarchy, we can at least show these sentences cannot have very very low complexity:

Lemma 7. *All true Σ_1 sentences are provable.*

Proof. Given any Σ_1 sentence φ, it is of the form

$$\exists x_1\cdots\exists x_k\psi(x_1,\cdots,x_k)$$

where $\psi(x_1,\cdots,x_k)$ is Δ_0. If φ is true, then there are natural numbers $n_1,\cdots,n_k$ such that $\psi(\overline{n_1},\cdots,\overline{n_k})$ is true. Since ψ is Δ_0, which only contains bounded quantification, PA — in fact, even a much weaker theory — will be able to prove it,

$$\mathsf{PA}\vdash\psi(\overline{n_1},\cdots,\overline{n_k})$$

Then according to our derivations rules, if we can derive instances of a formula we can of course derive the existential quantified ones,

$$\mathsf{PA}\vdash\exists x_1\cdots\exists x_k\psi(x_1,\cdots,x_k)$$

Hence, this shows PA also proves φ. □

We can now finally use this classification of arithmetical sentences to define a corresponding hierarchy of large complexity classes, through their definability by arithmetic formula:

Definition 5 (Definitional Complexity). *We say a subset $S \subseteq \mathbb{N}$ is Σ_n (resp. Π_n) if it is definable by a Σ_n (resp. Π_n) arithmetic formula. If a subset S is both Σ_n and Π_n, we say it is Δ_n.*

From the previous examples we have shown, the total set $\mathbb{N}$ and the empty set are obviously at the lowest of the hierarchy, since they are definable without any use of quantification. For prime numbers, it is easy to see that the previous formula we use can be slightly modified to the following Δ_0 formula,

$$\mathsf{prime}'(x) := \overline{2} \leq x \wedge \forall y < x(y \,|\, x \rightarrow y = \overline{1} \vee y = x)$$

In the previous formula $\mathsf{prime}(x)$ we define, the universal quantification there is unbounded; but in fact we only need to check those numbers that is less than x to test whether x is a prime or not. Hence, the set of prime numbers is also Δ_0. In the previous section, we have sketched a proof that every recursive subset is definable by some arithmetic formula. Here, we will in addition show the definitional complexity of recursive and r.e. subsets are Δ_1 and Σ_1, respectively. To do that, let's first observe the following lemma:

Lemma 8. *Every Δ_0 subset is computable by a TM.*

Proof. Suppose a subset S is definable by a Δ_0 formula $\varphi_S(x)$. Since $\varphi_S(x)$ only contains quantification bounded by x, given any $n \in \mathbb{N}$ we at most need to check n cases to decide the truth of $\varphi_S(\overline{n})$. Obviously, there exists a TM that does this job, hence compute S. □

Lemma 9. *A subset $S \subseteq \mathbb{N}$ is r.e. iff it is Σ_1.*

Proof. Suppose a subset S is Δ_1, viz. there exist a Σ_1 formula $\exists x \varphi_1(x, y)$ and a Π_1 formula $\forall x \varphi_2(x, y)$ that both define S. By definition, φ_1 and φ_2 are Δ_0, thus by Lemma 8 whether $\varphi_i(\overline{n}, \overline{m})$ is true is computable by a TM M_i, $i = 1, 2$. Then, we can construct a TM M, such that given input n it simulates the following computation: At the j-th step, it simulates M_i to check whether $\varphi_i(\overline{j}, \overline{n})$ is true; if at some point $\varphi_1(\overline{j}, \overline{n})$ is true then M accepts n; if at some point $\varphi_2(\overline{j}, \overline{n})$ is false then M rejects n. If $n \in S$, by definition there exists a number j_1 that $\varphi_1(\overline{j_1}, \overline{n})$ is true; if $n \notin S$, again by definition there exists j_2 that $\varphi_2(\overline{j_2}, \overline{n})$ is false. Hence, M must at some point halt and output the right answer. This shows that a Δ_1 set is recursive.

On the other hand, to prove every r.e. subset is Σ_1, we can use a similar strategy presented in the proof of Proposition 2. Given a suitable coding of TM, we can define a relation $\mathsf{Halt} \subseteq \mathbb{N}^3$, such that $\mathsf{Halt}(r, m, n)$ holds iff the TM coded by n halts within m steps when given the input r. Since the register states and transition functions of a TM are always finite, a careful analysis shows that $\mathsf{Halt}(r, m, n)$ must be definable by a Δ_0 formula $\varphi_H(x, y, z)$, since basically we only need to define a finite set and finite number of applications of a function between finite set; no quantification needs to be involved. Now for a r.e. set S, suppose it is the halting domain of a TM M with code n. For any r, by definition the following holds,

$$r \in S \Leftrightarrow \mathbb{N} \models \exists m \varphi_H(\overline{r}, m, \overline{n})$$

This shows that S is defined by the formula $\exists m \varphi_H(\overline{r}, m, \overline{n})$, which is Σ_1. □

Corollary 2. *A subset $S \subseteq \mathbb{N}$ is recursive iff it is Δ_1.*

Proof. By Lemma 3, the subset S is recursive iff both itself and its complement are r.e.. According to Lemam 9, a subset is r.e. iff it is definable by a Σ_1 formula. Thus, S is recursive iff both itself and its complement is Σ_1. Since if S is defined by φ, $\mathbb{N} \backslash S$ is defined by $\neg\varphi$. And because the negation of a Σ_1 formula is Π_1, S is recursive iff it is defined both by a Σ_1 and Π_1 formula, hence Δ_1. □

Corollary 3. *The set of theorems in* PA *is Σ_1.*

Proof. Proposition 1 shows that the set of (codes of) Th(PA) is r.e., thus by Lemma 9 Th(PA) is Σ_1. □

To illustrate how large these defined complexity classes are, let me compare Δ_0 sets and the other complexity class ELEM. ELEM is defined to contain the subsets that are computable by every k-exponential-time computable TM,

$$\mathsf{ELEM} := \bigcup_{k \in \mathbb{N}} k\text{-}\mathsf{EXP} = \mathsf{DTIME}\left(2^n\right) \cup \mathsf{DTIME}\left(2^{2^n}\right) \cup \cdots$$

The fact is, there exists a Δ_0 set that is not contained in ELEM. The reason for this is that we can define *hyper operations* (e.g. tetration) with Δ_0 formulas that goes faster than any level of exponentiation. However, these functions are still computable in an intuitive sense. Actually, by the Church-Turing thesis, the intuitively computable functions corresponds to recursive ones, viz. Δ_1 sets. Thus, Δ_1 can be roughly thought of all computable subsets without any resource restrictions. However, there are infinitely many larger complexity classes beyond Δ_1 according to the definitional complexity defined here. Furthermore, by Post's Theorem, each level of the definitional complexity hierarchy defined here is non-trivial, i.e. we do not have any collapses within the hierarchy, and the following holds for any $n \geq 1$,

$$\Delta_n \subsetneq \Sigma_n, \Delta_n \subsetneq \Pi_n,\ \Sigma_n \cup \Pi_n \subsetneq \Delta_{n+1}$$

Due to the scope of this essay, we cannot go any deeper into a more satisfactory analysis of Post's Theorem and its application for our defined hierarchy of complexity classes. We refer the reader to [Moschovakis, 1987] for a more detailed discussion of it.

Actually, after establishing the above results, we can also show how Gödel proved his First Incompleteness Theorem originally. The proof is very interesting in itself, because it proceeds by formalising a *paradox* in the formal language of arithmetic. The involved paradox is the liar paradox: the sentence "This sentence is false." cannot be consistenly assigned a truth value. Furthermore, Gödel's original proof helps us to show the tremendous complexity of the set $\mathbb{N}(\mathcal{L})$, which will be discussed soon.

Let us use $\ulcorner e \urcorner$ to denote the numeral of the code of e, viz. $\overline{\alpha(e)}$, in $\mathcal{L}$. To illustrate Gödel's original proof, we need to first show a diagonal lemma which allows Gödel to formalise the liar paradox.

Lemma 10 (Diagonal Lemma). *For any formula $\varphi(x)$ in $\mathcal{L}$, we can efficiently construct a sentence γ such that the following holds,*

$$\mathsf{PA} \vdash \gamma \leftrightarrow \varphi(\ulcorner \gamma \urcorner)$$

Proof. Let us define the relation Sub to represent substitution, i.e. for any formula $\varphi(x)$ and any expression e,

$$\mathrm{Sub}(\alpha(\varphi(x)), \alpha(e), n) \text{ holds} \Leftrightarrow n = \alpha(\varphi(e))$$

Obviously, Sub is a recursive relation, thus can be defined by a Δ_1 formula $\mathsf{sub}(x, y, z)$. We then define $\theta(x)$ to be the following formula,

$$\theta(x) := \exists y(\mathsf{sub}(x, x, y) \wedge \varphi(y))$$

and γ to be the following sentence,

$$\gamma := \theta(\ulcorner \theta(x) \urcorner)$$

First, notice that

$$\mathsf{PA} \vdash \mathsf{sub}(\ulcorner \theta(x) \urcorner, \ulcorner \theta(x) \urcorner, \ulcorner \gamma \urcorner)$$

By our defnition of γ we indeed have that

$$\mathrm{Sub}(\alpha(\theta(x)), \alpha(\theta(x)), \alpha(\gamma)) \text{ holds}$$

Hence, $\mathsf{sub}(\ulcorner \theta(x) \urcorner, \ulcorner \theta(x) \urcorner, \ulcorner \gamma \urcorner)$ is true; since sub is also Δ_0, this shows it is also provable. This would then imply that

$$\mathsf{PA} \vdash \varphi(\ulcorner \gamma \urcorner) \rightarrow \gamma$$

However, since the relation Sub is function-like, we would also have

$$\mathsf{PA} \vdash \forall y(\mathrm{sub}(\ulcorner\theta(x)\urcorner, \ulcorner\theta(x)\urcorner, y) \rightarrow y = \ulcorner\gamma\urcorner)$$

Combining the two it follows then

$$\mathsf{PA} \vdash \gamma \leftrightarrow \varphi(\ulcorner\gamma\urcorner)$$

This proves the diagonal lemma. □

With the help of the diagonal lemma, Gödel's original proof of the First Incompleteness Theorem can the proceed as follows: According to the proof of Lemma 1, we have recursive relation $\mathsf{Prf}(m, n)$ that represents whether m is the code of a proof of the formula coded by n. Then by Corollary 2, there is a Δ_1 formula $\mathrm{prf}(x, y)$ in $\mathcal{L}$ that represents this relation. We can then define a *provability predicate* $\mathrm{Bew}(x)$ as follows,[4]

$$\mathrm{Bew}(x) := \exists y \mathrm{prf}(y, x)$$

Notice that $\mathrm{Bew}(x)$ would be Σ_1. By the diagonal lemma, there exists a sentence γ such that the following is provable,

$$\mathsf{PA} \vdash \gamma \leftrightarrow \neg \mathrm{Bew}(\ulcorner\gamma\urcorner)$$

Intuitively, γ is a sentence saying it is provable iff it is not provable, which is the formalisation of the liar paradox using provability in the first-order language of arithmetic. We can actually show γ is then a true but unprovable sentence, provided that PA is consistent, i.e. no contradiction can be deduced from PA. The reasoning goes as follows: If γ is provable, then $\neg \mathrm{Bew}(\ulcorner\gamma\urcorner)$ is also provable, thus true, which by our definition means there are some number m such that $\mathrm{prf}(\overline{m}, \ulcorner\gamma\urcorner)$ is true. However, since prf represents Prf, this would imply that there actually is a proof of γ, contradictory to our assumption. Hence, γ is not provable. However, since γ is logically equivalent to $\neg \mathrm{Bew}(\ulcorner)\urcorner$, which says γ is not provable, γ would then be true, since it is indeed not provable. This then shows that γ is a true but unprovable sentence.

Gödel's construction is really interesting and remarkable — it uses the idea of a paradox and gives us an explicitly example of how to find a true but unprovable sentence, which arguably cannot easily be found using purely complexity-theoretic methods. But what is even more remarkable is that the above process can be also used to show that *truth is not definable* in arithmetic:

Theorem 3 (Undefinability of Truth). *There is no arithmetic formula $\chi(x)$ that defines the set $\mathbb{N}(\mathcal{L})$ (under the coding α).*

[4] The "Bew" in $\mathrm{Bew}(x)$ stands for "bewis", which is the German for "proof".

Proof. Suppose there is one formula $\chi(x)$ that defines truth, i.e. for any sentence ψ the following holds,

$$\psi \in \mathbb{N}(\mathcal{L}) \Leftrightarrow \mathbb{N} \models \chi(\ulcorner\psi\urcorner)$$

Now according to the diagonal lemma, there exists a sentence γ such that

$$\mathsf{PA} \vdash \gamma \leftrightarrow \neg\chi(\ulcorner\gamma\urcorner)$$

We show that γ can neither be true nor be false. If γ is true, then $\chi(\ulcorner\gamma\urcorner)$ is false by the diagonal property, which further implies γ cannot be true due to the fact that $\chi(x)$ defines truth. A similar argument shows that γ cannot be false, either. Thus, there cannot be any formula that defines truth. □

The above proof basically uses the fact that if indeed truth can be defined, then due to the diagonal lemma the real liar paradox — what Gödel previously defined in his proof of the First Incompleteness Theorem is the version where truth is substituted by provability — can be formalised in arithmetic, that generates a real paradox.

Corollary 3 has shown that the complexity of provable sentences $\mathrm{Th}(\mathsf{PA})$ is Σ_1, which is at the very bottom of our hierarchy. However, Theorem 3 tells us that the set $\mathbb{N}(\mathcal{L})$ cannot be defined by any arithmetic sentence, which means the complexity of the set $\mathbb{N}(\mathcal{L})$ is unmeasurable by the infinite, non-collapsing hierarchy. $\mathbb{N}(\mathcal{L})$ is beyond Σ_n or Π_n, for any possible $n \in \mathbb{N}$! The contrast shows us that what we can prove in our formal system is significantly less complex than what is true.

However, that does not mean we cannot say anything about the complexity of $\mathbb{N}(\mathcal{L})$. There are frameworks that extended arithmetic hierarchy that incorporates sets like $\mathbb{N}(\mathcal{L})$ and gives them a precise definition of their definitional complexity. This more general framework is call *analytical hierarchy*, which involves the use of second-order arithmetic. The complexity of $\mathbb{N}(\mathcal{L})$ is at the lowest level of complexity in the analytical hierarchy that beyond arithmetic hierarchy. Another direction to go is to look at various *relative hierarchies*. For each subset $S \subseteq \mathbb{N}$, we can define Σ_n^S and Π_n^S sets relative to S, where S acts like an oracle. There given some non-trivial subsets as an oracle, we will find $\mathbb{N}(\mathcal{L})$ appearing at some level in the relativised arithmetic hierarchy. However, we will not further explain in more detail.

Let us zoom out a bit to see what we have achieved in the previous two sections. In Section 2, we have used a recursion theoretic framework to prove the First Incompleteness Theorem given by Gödel. In Section 3, we have connected the recursion theoretic notions with arithmetic hierarchy, and have used definitional complexity to classify a corresponding hierarchy of large complexity classes. We've shown that the complexity of the set of theorems of PA, which is Σ_1, is significantly lower than the complexity of $\mathbb{N}(\mathcal{L})$, which is beyond all the arithmetic level. This helps us gain insight of the fundamental ability of generating mathematical proofs through our first-order proof system.

What is also interesting is the way we define the notion of definitional complexity. Not like the usual characterisation of complexity classes by their resources that takes to compute the results, this notion of complexity is defined by how complex it is to *describe* something. This hints at another way of looking at complexity at a whole, which is precisely what Kolmogorov complexity provides us. As we'll see in the next section, Kolmogorov complexity has lead to much greater insights in our understanding of the ability and limits of our formal system, and really makes it clear that foundation of mathematics is in such a close tie with complexity theory.

4 Kolmogorov Complexity & Foundation of Mathematics

Kolmogorov complexity was initially introduced independently, with different motivations from different perspectives, in [Raymond, 1964], [Kolmogorov, 1965], and [Chaitin, 1966]. After more than 50 years of development it has now become a mature area of active research. As hinted at the end of previous section, Kolmogorov complexity concerns the difficulty of *describing* an object. Since the subject of Kolmogorov complexity is not standardly studied in complexity theory, here we give a brief introduction to it. For a more complete but still introductory treatment, see [Grünwald and Vitányi, 2008]; or for a comprehensive textbook, see [Li et al., 2008]. The main goal of this section is to show that with the perspective of Kolmogorov complexity, for every formal system there is an essential complexity barrier that fundamentally restricts the ability of what can be proved within that system. This provides us yet another and deeper understanding of how complexity theory is at the heart of foundation of mathematics.

Actually, the Kolmogorov complexity can be defined for any universal programming language that are sufficient to implement a universal TM. With respect to such a language L, we can define the Kolmogorov complexity $K_L(x)$ for a string x to be the length of the shortest programme in L that outputs x. Here we choose to work directly in the computational model provided by TM. Let us for now fix a universal TM U. The Kolmogorov complexity K_U (with respect to the choice of U) will be understood as a function from natural numbers to natural numbers,

$$K_U : \mathbb{N} \longrightarrow \mathbb{N}$$

under the identification of natural numbers as binary strings under their lexicographic order, i.e. we have the following correspondence,

$$\lambda \mapsto 0, \ 0 \mapsto 1, \ 1 \mapsto 2, \ 00 \mapsto 3, 01 \mapsto 4, \cdots$$

From now on, we will no longer distinguish between natural numbers and binary strings. We let l be the length function on binary strings. Under our identification, we have $l(x) = \lfloor \log(x+1) \rfloor$, for $x \geq 2$.

The function K_U is defined as follows: given any $n \in \mathbb{N}$, the Kolmogorov complexity of n is the shortest input that outputs n,

$$K_U(n) = \min_{y \in \mathbb{N}} \{ \, l(y) \mid U(y) = n \, \}$$

We will also call an input y of a universal TM as a programme of U.[5] Notice that for any n, there exists a TM M that simply remembers the string n and prints it out given any input. Since U is universal, it follows that there must be a programme y making U simulates M and output n. Hence, K_U as a function is well-defined.

It does not really matter which universal TM we have chosen, because it can be easily shown that for any two such choices U_1 and U_2, for any $n \in \mathbb{N}$ we would have

$$|K_{U_1}(n) - K_{U_2}(n)| \leq c$$

for some constant c not depending on n. This is simply because any universal TM is able to simulate one another. For any programme y that runs on U_2, we only need to add a constant length of programme y_0 to it, telling U_1 to simulates U_2, and we can then run y and it would obtain the same output. The added part y_0 of the programme is to let U_1 simulate U_2, which does not depend on any specific input n. Hence, the difference between $K_{U_1}(n)$ and $K_{U_2}(n)$ will be bounded by a constant. Hence from now on, we will omit the subscript and simply write $K(n)$ as the Kolmogorov complexity of n, implicitly assuming a fixed universal TM U.

From the general idea of previous section, we would like to see whether Kolmogorov complexity is definable by any arithmetic formula, and if it is, what's it definitional complexity. The following lemma first shows that the binary relation represented by $y < K(n)$ is definable, and is Π_1:

Lemma 11. *There exists a Π_1 formula $\mathsf{k}(u, v)$ defining the relation $y < K(n)$.*

Proof. Let U be our fixed universal TM. We first show that the relation represented by $U(y) = n$ is r.e.. We can simply construct a TM M, such that when given the input (y, n) it simulates U on the input y. If U ever halts and outputs n, we let M accepts; if the output is not n, we let M rejects; and if U runs forever then M will also runs forever. By our definition, this shows that the relation represented by $U(y) = n$ is r.e.. Now by Lemma 9, there exists a Σ_1 formula $\varphi_U(u, v)$ that defines the relation represented by $U(y) = n$. Now obviously, the relation $y < K(n)$ can be defined by the following formula,

$$\mathsf{k}(u, v) := \forall z \leq u(\neg \varphi_U(z, v))$$

[5] The idea that numbers are themselves also programmes started very early on in the beginning of computability theory. Such structures with each element in a set can be considered as partial functions on the set itself form a very important class of models of computation; see [Van Oosten, 2008] for more on this.

which is obviously Π_1, because the negation of a Σ_1 formula $\varphi_U(z, v)$ is Π_1, and adding a bounded universal quantifier in front does not change this fact. □

Using this formula $\mathsf{k}(u, v)$ in our language of arithmetic, the first thing we can show is that it helps to find a whole other class of true but unprovable sentences, which to a great extent generalises the incompleteness results. The following proof was first due to Chaitin.

Theorem 4 (Chaitin's Incompleteness Theorem). *There exists a constant c such that for any n with Kolmogorov complexity greater than c, viz. $c < K(n)$, the sentence $k(\overline{c}, \overline{n})$ is not provable.*

Proof. The idea is to find a large enough constant c, such that though the Kolmogorov complexity of n is indeed greater than c, it will never be provably so. Let the fixed universal TM be U. We find a programme (a number) m such that when it runs on U it performs the following task,

$$U(m) \rightsquigarrow \text{find the least } x \text{ that } \mathsf{Prf}(x, \alpha(\mathsf{k}(\overline{c}, \overline{n})) \text{ for any } n\text{, print } n$$

In other words, the programme m running on U makes it to find the shortest proof of a sentence of the form $\mathsf{k}(\overline{c}, \overline{n})$; if there is such a proof and such a n, one then print out n. Since we've shown in Lemma 1 that the theorems, as well as proofs, in PA is r.e., if there exists such a proof then $U(m)$ will halt at some finite number of steps; otherwise it will run forever. We now find a pair of m, c such that $m < c$. This is certainly doable because the programme size of m that doing this task grows at the order of $\log c$.

We now show that $U(m)$ never halts. Suppose it finds some n such that there is a proof p of $\mathsf{k}(\overline{c}, \overline{n})$, and prints n out. By the soundness lemma this must be true, hence indeed we have the Kolmogorov complexity of n is greater than c. However, we just used a programme m, which has size less than c, that also prints out n; this implies the Kolmogorov complexity of n must be less than or equal to m, contradictory. Thus, $U(m)$ can never halt, and thus there can be no proof of $\mathsf{k}(\overline{c}, \overline{n})$. □

However, even though there are no proof of $\mathsf{k}(\overline{c}, \overline{n})$ for any n, there are infinitely many numbers that indeed has Kolmogorov complexity greater than c. This follows from a simple counting argument, because the number of programmes less than length c is at most 2^c, which is a finite number. Thus, Theorem 4 has found us infinitely many true but unprovable sentences in first-order arithmetic, which greatly extends the original Gödel's construction. It is also interesting to notice that the original Gödel's construction, which is a sentence γ provably equivalent to $\neg \mathrm{Bew}(\ulcorner \gamma \urcorner)$, is actually Σ_2. This is because $\mathrm{Bew}(x)$ is a Σ_1 predicate, thus $\neg \mathrm{Bew}(x)$ is Π_1. By the construction presented in the diagonal lemma above there is an additional existential quantifier added in γ, thus γ is Σ_2. However, by Lemma 11 the true but unprovable sentences constructed here is only Π_1. This shows us that incompleteness phenomena already happen at a quite low complexity level.

The constant c represents an inherent complexity constraint of our formal system. Due to the expressive power and strength of different formal systems, the constant c will change accordingly. However, for sufficiently strong systems, like in our case PA, it will remain roughly at the same magnitude. It is also instructive to actually see what quantitative value of such a constant c might be. Here it is more convenient to work in a universal programming language, than in our case of a universal TM, because different coding schemes of TM will probably give us quite different results about the value of c. In algorithmic complexity theory, the most commonly used language is perhaps LISP, e.g. see [Chaitin, 1996]. There, Chaitin gives some explicit codes that arguably shows that such a constant c will only be a few thousand digits, which is a very low number compared to our current normal data size!

5 Conclusion

In this essay, we have set out the task of investigating the relationship between formal mathematical systems and complexity theory. As mentioned, formal systems, including our primary focus of first-order Peano arithmetic or the much more powerful axiomatic set theory, are introduced to set a solid yet powerful foundation of mathematics. However, a closer analysis at them shows us that there are fundamental complexity-theoretic constraints on the ability of any sufficiently strong, consistent formal systems. Section 2 and Section 3 have used recursion theory and definitional complexity to explore how to understand the Gödel's First Incompleteness Theorem in a more conceptual way. Section 4 has further extended the tie between formal system and complexity by showing how Kolmogorov complexity sets up an essential complexity barrier for any sufficiently strong formal system. These results have shown us that complexity theory is at the heart of foundation of mathematics.

Let me conclude the essay with the following line from Borges' "Fragmentos de un Evangelio apócrifo", which I first read in [Hirschfeldt, 2015]:

> *Nada se edifica sobre la piedra, todo sobre la arena, pero nuestro deber es edificar como si fuera piedra la arena.* [Nothing is built on stone, all on sand, but our duty is to build as if the sand were stone.]

Indeed, mathematics is not built on stone, but on sand. Modern mathematical logic and complexity theory are our first steps towards realising this, and starting to seriously face the task of building mathematics on sand, rather than stone.

References

[Chaitin, 1966] Chaitin, G. J. (1966). On the length of programs for computing finite binary sequences. *Journal of the ACM (JACM)*, 13(4):547–569.

[Chaitin, 1987] Chaitin, G. J. (1987). *Program Size, Halting Probabilities, Randomness, & Metamathematics*, page 91–91. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

[Chaitin, 1990] Chaitin, G. J. (1990). *Information, randomness & incompleteness: papers on algorithmic information theory*, volume 8. World Scientific.

[Chaitin, 1996] Chaitin, G. J. (1996). The limits of mathematics. *J. UCS*, 2(5):270–305.

[Franks, 2009] Franks, C. (2009). *The autonomy of mathematical knowledge: Hilbert's program revisited.* Cambridge University Press.

[Gödel, 1931] Gödel, K. (1931). Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik*, 38(1):173–198.

[Grünwald and Vitányi, 2008] Grünwald, P. D. and Vitányi, P. M. (2008). Algorithmic information theory. *Handbook of the Philosophy of Information*, pages 281–320.

[Hirschfeldt, 2015] Hirschfeldt, D. R. (2015). *Slicing the truth: On the computable and reverse mathematics of combinatorial principles.* World Scientific.

[Kikuchi, 1997] Kikuchi, M. (1997). Kolmogorov complexity and the second incompleteness theorem. *Archive for Mathematical Logic*, 36(6):437–443.

[Kleene, 1952] Kleene, S. C. (1952). *Introduction to metamathematics.* North Holland.

[Kolmogorov, 1965] Kolmogorov, A. N. (1965). Three approaches to the quantitative definition ofinformation'. *Problems of information transmission*, 1(1):1–7.

[Kritchman and Raz, 2010] Kritchman, S. and Raz, R. (2010). The surprise examination paradox and the second incompleteness theorem. *Notices of the AMS*, 57(11):1454–1458.

[Kunen, 2014] Kunen, K. (2014). *Set theory an introduction to independence proofs.* Elsevier.

[Li et al., 2008] Li, M., Vitányi, P., et al. (2008). *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer.

[Moschovakis, 1987] Moschovakis, Y. (1987). *Descriptive Set Theory.* ISSN. Elsevier Science.

[Peckhaus, 1997] Peckhaus, V. (1997). *Logik, Mathesis universalis und allgemeine Wissenschaft. Leibniz und die Wiederentdeckung der formalen Logik im 19. Jahrhundert.* Akademie-Verlag (Logica Nova), Berlin.

[Raatikainen, 1998] Raatikainen, P. (1998). On interpreting chaitin's incompleteness theorem. *Journal of Philosophical Logic*, 27(6):569–586.

[Raymond, 1964] Raymond, J. S. (1964). A formal theory of inductive inference. *Information and Control*, 7:1–22. part 1 and part 2.

[Rogers, 1987] Rogers, H. (1987). *Theory of Recursive Functions and Effective Computability.* MIT Press, Cambridge, MA, USA.

[Tarski et al., 1953] Tarski, A., Mostowski, A., and Robinson, R. M. (1953). *Undecidable theories*, volume 13. Elsevier.

[Van Oosten, 2008] Van Oosten, J. (2008). *Realizability: an introduction to its categorical side.* Elsevier.

[Wang, 1997] Wang, H. (1997). *A logical journey: From Gödel to philosophy.* MIT Press.

[Zach, 2007] Zach, R. (2007). Hilbert's program then and now. In *Philosophy of logic*, pages 411–447. Elsevier.