Introduction

Gödel's incompleteness theorems are among the most important results in mathematical logic. In order to fully appreciate their significance, it is necessary to explain the historical background. At the turn from the 19th to the 20th century, several paradoxes surfaced in the foundations of mathematics, leading to increasing uncertainty concerning the solidity of these foundations. There have been a number of reactions to that situation, the most far-reaching of them was Hilbert's.

At the beginning of the 1920ies, Hilbert put forward a proposal for the foundations of mathematics which is now called "Hilbert's programme". This programme is based on a simple but striking observation which underlies all formalisation efforts, in particular also Russel and Whitehead's Principia Mathematica: in mathematics we talk about infinite sets, real numbers, real-valued functions, operators transforming such functions, etc. in short: about abstract, infinite objects. However, we do so in an inherently finite way; every proof is a finite sequence of symbols, taken from some finite set, every theory is a finite succession of such proofs. What we say and prove about such objects is thus inherently finite. For Hilbert, the part of mathematics which deals with elementary properties of finite sequences of symbols was relying only on a purely intuitive basis. Their elementary properties and relations are immediate and not mediated by logic. Therefore they are not susceptible to the possibility of a contradiction. Elementary statements about such sequences thus form a secure basis for the foundations of mathematics. Hilbert proposed to use this basis for giving an axiomatic formalisation of all of mathematics and to prove this formalisation consistent, i.e., to show that no contradiction can arise based on consideration of finite sequences of symbols alone. Thus, so Hilbert thought, one could justify the use of abstract concepts in mathematics.

However, this hope was shattered by Gödel's incompleteness theorems, which were published in 1931. Informally, they can be stated as follows:

Theorem (First Incompleteness Theorem)**.** *Let T be a consistent and axiomatisable theory "containing arithmetic", then there is a sentence σ s.t. $T \nvdash \sigma$ and $T \nvdash \neg\sigma$.*

Theorem (Second Incompleteness Theorem)**.** *Let T be a consistent and axiomatisable theory "containing arithmetic", then $T \nvdash \mathsf{Con}_T$.*

Without explaining these statements in detail, let us just note that the conditions imposed on T in these two theorems are not identical but, in both cases, encompass all situations envisaged by Hilbert in his programme to prove consistency statements. The second incompleteness theorem clearly destroys Hilbert's programme, for if a theory cannot prove its own consistency, then an even weaker theory, for example one that speaks only about finite sequences of symbols, cannot prove it either. Thus, after publication of the incompleteness theorems, Hilbert's programme had to be given up.

Nevertheless, the investigation of the logical foundations of mathematics that has been carried

out since, while not leading to consistency proofs as envisaged by Hilbert, has led to an improvement of our understanding which was sufficient for dissipating doubts about the consistency of mathematical reasoning. Gödel's incompletess theorems have become a cornerstone of logic (in mathematics, philosophy, and computer science). The proof techniques introduced by Gödel in these results, arithmetisation (also called "Gödelisation") in conjunction with diagonalisation, have become central for many results in mathematical logic.

This course is designed as a second course in mathematical logic, centered around the incompleteness theorems. We are assuming passive and active knowledge of first-order logic, in particular, the syntax and semantics of formulas, proof calculi, models, and the completeness theorem. We will take the incompleteness theorems as central aims of this course. However, we will not proceed there in the most direct way possible. Instead, we take them as occasion to study important notions and results surrounding them, in particular, in computability theory and formal theories of arithmetic.

As further literature, [4] can be recommended as a compact presentation of the incompleteness theorems and [1] as a comprehensive reference on theories of arithmetic. Furthermore, [2] provides a more model-theoretic perspective on theories of arithmetic and [5, 3] are useful for background in computability theory. These lecture notes owe a debt to all of these sources.

Bibliography

[1] Petr Hájek and Pavel Pudlák. *Metamathematics of First-Order Arithmetic*. Springer, 1993.

[2] Richard Kaye. *Models of Peano Arithmetic*, volume 15 of *Oxford Logic Guides*. Clarendon Press, 1991.

[3] Piergiorgio Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., 1989.

[4] Craig Smorynski. The Incompleteness Theorems. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 821–865. North-Holland, 1977.

[5] Robert I. Soare. *Recursively Enumerable Sets and Degrees*. Perspectives in Mathematical Logic. Springer, 1987.

Chapter 1

Computability

Computability theory is, along with proof theory, set theory, and model theory, one of the four main areas of mathematical logic. The incompleteness theorems are strongly connected, both historically and mathematically, to central notions and techniques of computability theory. We will therefore start this couse on the former with a brief introduction to the latter. The aim of this chapter is to prove the existence of a recursively enumerable but undecidable set. From this result we will soon be able to obtain a weak version of the first incompleteness theorem as a corollary. As we go along, we pick up some notions, in particular concerning coding, also called arithmetisation or "Gödelisation", that will be useful later on.

1.1 The partial recursive functions

One approach to defining the set of functions which are computable in the intuitive sense is to start "from below": define some functions which are obviously computable, then define closure operators which transform computable functions in computable functions. We will follow this approach here.

Definition 1.1. The *basic functions* are:

1. the constant (nullary function) $0 \in \mathbb{N}$,
2. the *successor function* $\mathrm{S} : \mathbb{N} \to \mathbb{N}, x \mapsto x + 1$,
3. for all $k \geq 1, 1 \leq i \leq k$, the *projection function* $\mathrm{P}_i^k : \mathbb{N}^k \to \mathbb{N} : (x_1, \ldots, x_k) \mapsto x_i$.

All of the basic functions are obviously computable.

Definition 1.2. Let $f : \mathbb{N}^n \to \mathbb{N}$, $g_1 : \mathbb{N}^k \to \mathbb{N}, \ldots, g_n : \mathbb{N}^k \to \mathbb{N}$. Then the function obtained by *composition* of f with $g_1, \ldots, g_n$ is

$$\mathrm{Cn}[f, g_1, \ldots, g_n] : \mathbb{N}^k \to \mathbb{N}, \overline{x} \mapsto f(g_1(\overline{x}), \ldots, g_n(\overline{x})).$$

If $n = 1$, then $\mathrm{Cn}[f, g]$ is usually written as $f \circ g$. If $f, g_1, \ldots, g_n$ are computable, then so is h: in order to compute h, we first compute $y_i = g_i(\overline{x})$ for $i = 1, \ldots, n$ which is possible by assumption and then we compute $f(y_1, \ldots, y_n)$ which is, again, possible by assumption. Another way to put the above definition is to say that, for $k, n \in \mathbb{N}$, Cn_n^k is an operator, transforming functions into functions, i.e., C_n^k is of type $(\mathbb{N}^n \to \mathbb{N}) \times (\mathbb{N}^k \to \mathbb{N})^n \to (\mathbb{N}^k \to \mathbb{N})$.

Definition 1.3. Let $f : \mathbb{N}^k \to \mathbb{N}$ and $g : \mathbb{N}^{k+2} \to \mathbb{N}$. Then the function obtained by *primitive recursion* of f and g is $\Pr[f, g] = h : \mathbb{N}^{k+1} \to \mathbb{N}$ defined by

$$\begin{aligned} h(\overline{x}, 0) &= f(\overline{x}), \text{ and} \\ h(\overline{x}, y+1) &= g(\overline{x}, y, h(\overline{x}, y)). \end{aligned}$$

If f and g are computable then so is h. Let $\overline{x} \in \mathbb{N}^k$. We argue, informally, by induction on $y \in \mathbb{N}$: if $y = 0$ then, by assumption, $f(\overline{x})$ can be computed and thus $h(\overline{x}, y)$ can be. If $y > 0$, say $y = y' + 1$, we can compute $z = h(\overline{x}, y')$ by induction hypothesis and then $h(\overline{x}, y) = g(\overline{x}, y', z)$ from it by assumption.

Definition 1.4. A function $f : \mathbb{N}^k \to \mathbb{N}$ is called *primitive recursive* if it can be obtained from the basic functions by a finite number of applications of the operators composition and primitive recursion.

Example 1.1. Consider the functions $f = \mathrm{P}^1_1 : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N}^3 \to \mathbb{N}, (x, y, z) \mapsto z + 1$. Then $g = \mathrm{S} \circ \mathrm{P}^3_3$. By primitive recursion of f and g we obtain the function $h : \mathbb{N}^2 \to \mathbb{N}$ defined by

$$\begin{aligned} h(x, 0) &= \mathrm{P}^1_1(x) = x, \text{ and} \\ h(x, y+1) &= g(x, y, h(x, y)) = h(x, y) + 1. \end{aligned}$$

In other words, h is the addition of natural numbers which is hence primitive recursive. This fact can also be written as $+ = \Pr[\mathrm{P}^1_1, \mathrm{Cn}[\mathrm{S}, \mathrm{P}^3_3]]$.

Lemma 1.1. *The following functions are primitive recursive*

1. *addition* $(x, y) \mapsto x + y$,
2. *the constant function* $\mathrm{c}^k_z : \mathbb{N}^k \to \mathbb{N}, (x_1, \ldots, x_k) \mapsto z$,
3. *multiplication* $(x, y) \mapsto x \cdot y$
4. *truncated predecessor* $x \mapsto \mathrm{p}(x) = \begin{cases} 0 & \text{if } x = 0 \\ x - 1 & \text{if } x > 0 \end{cases}$
5. *truncated subtraction* $(x, y) \mapsto x \dot{-} y = \begin{cases} 0 & \text{if } x \leq y \\ x - y & \text{if } x > y \end{cases}$
6. *the characteristic function of less than or equal* $(x, y) \mapsto \chi_{\leq}(x, y) = \begin{cases} 1 & \text{if } x \leq y \\ 0 & \text{if } x > y \end{cases}$
7. *the characteristic function of equality* $(x, y) \mapsto \chi_{=}(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$

Proof. 1. has been shown in Example 1.1. For showing 2., first note that $\mathrm{c}^0_z = \mathrm{Cn}[\mathrm{S}, \mathrm{Cn}[\mathrm{S} \cdots \mathrm{Cn}[\mathrm{S}, 0] \cdots]]$. For $k = 1$ we use a trick based on the Pr-operator and define $\mathrm{c}^1_z = \Pr[\mathrm{c}^0_z, \mathrm{P}^2_2]$. Then $\mathrm{c}^1_z(0) = \mathrm{c}^0_z = z$ and $\mathrm{c}^1_z(y+1) = \mathrm{P}^2_2(y, \mathrm{c}^1_z(y)) = \mathrm{c}^1_z(y) = z$. For $k \geq 2$ we can simply define $\mathrm{c}^k_z = \mathrm{Cn}[\mathrm{c}^1_z, \mathrm{P}^k_1]$. For 3. consider that $x \cdot 0 = 0$ and $x \cdot (y+1) = x \cdot y + x$, i.e., $\cdot = \Pr[f, g]$ where $f(x) = 0$ and $g(x, y, z) = z + x$, i.e., $f = \mathrm{c}^1_0$ and $g = \mathrm{Cn}[+, \mathrm{P}^3_3, \mathrm{P}^3_1]$. For 4. we can simply define $\mathrm{p} = \Pr[0, \mathrm{P}^2_1]$. For 5. we use a primitive recursive definition based on $x \dot{-} 0 = x$ and $x \dot{-} (y+1) = \mathrm{p}(x \dot{-} y)$. For 6. observe that $\chi_{\leq}(x, y) = 1 \dot{-} (x \dot{-} y)$. For 7. note that $\chi_{=}(x, y) = \chi_{\leq}(x, y) \cdot \chi_{\leq}(y, x)$. □

At this point one may start to wonder: are the primitive recursive functions all computable functions? did we miss some? The following informal argument shows that there are computable functions which are not primitive recursive. Every primitive recursive function can be defined by a finite string of symbols from a fixed alphabet. Thus all such definitions can be effectively listed. Let f_n be the n-th function in that list and define $g(n) = f_n(n) + 1$. Then g cannot be in this list, for suppose it were, i.e., $g = f_e$, then $g(e) = f_e(e) = f_e(e) + 1$, contradiction. So g is not primitive recursive. However, g is computable in the intuitive sense. This kind of argument, diagonalisation, will reappear at several central places in this course. This argument applies to every set of total functions which can be effectively enumerated. However, diagonalisation is not an obstacle for partial functions, since $f_e(e)$ may simply be undefied. This motivates the following considerations.

Definition 1.5. A *partial function* from $\mathbb{N}^k$ to $\mathbb{N}$, in symbols $f : \mathbb{N}^k \hookrightarrow \mathbb{N}$, is a function $f : D \to \mathbb{N}$ for some $D \subseteq \mathbb{N}^n$.

If $\overline{x} \in D$, we say that f *is defined on* $\overline{x}$ and write $f(\overline{x}) \downarrow$. Analogously, if $\overline{x} \in \mathbb{N}^k \setminus D$, we say that f *is not defined on* $\overline{x}$, in symbols: $f(\overline{x}) \uparrow$. If, for a partial function $f : \mathbb{N}^k \hookrightarrow \mathbb{N}$ and a $k \in \mathbb{N}$, we write $f(\overline{x}) = k$ this includes $f(\overline{x}) \downarrow$. Similarly, given a second partial function $g : \mathbb{N}^k \hookrightarrow \mathbb{N}$, if we write $f = g$, then this includes both the statement that the domain of g is equal to that of f and that f and g have the same value on every element of their domain. The definitions of composition and primitive recursion generalise naturally to partial functions (where a result of a function is only defined if all results required for computing it by the respective operator are defined).

Example 1.2. If $f : \mathbb{N} \hookrightarrow \mathbb{N}, x \mapsto \begin{cases} \frac{x}{2} & \text{if } x \text{ is even} \\ \text{undefined} & \text{otherwise} \end{cases}$ and $g : \mathbb{N} \hookrightarrow \mathbb{N}$ is defined by $g = \mathrm{Cn}[\cdot, \mathrm{c}_0^1, f]$, then $g(x) = \begin{cases} 0 & \text{if } x \text{ is even} \\ \text{undefined} & \text{otherwise} \end{cases}$.

In all programming languages there are constructs that allow to start a recursion or an iteration *without* knowing in advance how often it will be repeated. Instead a condition is given which decides when to terminate the recursion/iteration, for example `while`- or `repeat ... until`-loops in imperative programming languages. Functions defined using such loops are clearly computable in the intuitive sense. However, in such constructs we do not have a guarantee that the condition will eventually be met, the computation may not terminate. In case of non-termination the value of the function that is computed is not defined. In our setting of operator terms, this behviour is modelled with the minimisation operator.

Definition 1.6. Let $f : \mathbb{N}^{k+1} \hookrightarrow \mathbb{N}$, then the function obtained from *minimisation* of f is $\mathrm{Mn}[f] = g : \mathbb{N}^k \hookrightarrow \mathbb{N}$, defined as

$$g(\overline{x}) = \begin{cases} y & \text{if } f(\overline{x}, y) = 0 \text{ and } \forall y' < y \, f(\overline{x}, y') \downarrow \text{ and } f(\overline{x}, y') \neq 0 \\ \text{undefined} & \text{if there is no such } y \end{cases}.$$

If f is computable, then so is g: we compute g by computing $f(\overline{x}, 0), f(\overline{x}, 1), \ldots$ until we find a y with $f(\overline{x}, y) = 0$. If one of the computations $f(\overline{x}, y')$ does not terminate, then the computation of g does not terminate. If all the computations of $f(\overline{x}, y')$ terminate but none of them yields 0, then the computation of g does not terminate.

We will often use the following notation: for an $f : \mathbb{N}^{k+1} \hookrightarrow \mathbb{N}$ we write $\mu y \, f(\overline{x}, y)$ for the

function

$$\overline{x} \mapsto \begin{cases} \text{the smallest } y \text{ s.t. } f(\overline{x}, y) = 1 \text{ and } f(\overline{x}, y') = 0 \text{ for all } y' < y & \text{if such a } y \text{ exists} \\ \text{undefined} & \text{otherwise} \end{cases}$$

In particular, this notation will be useful if f is the characteristic function of a relation R, $\mu y\, \chi_R(\overline{x}, y)$ is the smallest y s.t. $R(\overline{x}, y)$, if there exists one. Because of this notation, Mn is often also referred to as μ-recursion.

Definition 1.7. A *partial recursive function* is a partial function $f : \mathbb{N}^n \hookrightarrow \mathbb{N}$ that can be obtained from the basic functions by a finite number of applications of the operators of composition, primitive recursion, and minimisation.

A *recursive function* is a partial recursive function which is total.

At this point we can pause again to ask whether we have characterised the set of computable functions (by the set of partial recursive functions). It is now important to observe that this statement cannot be proven mathematically since the notion "computable (in the intuitive sense)" is not mathematical. However, there exists a large number of formalisms for modelling computation which are based on different paradigms for machines or programs which all turn out to be equivalent in the sense that they can compute exactly the partial recursive functions. This situation has led to the *Church-Turing thesis*: a partial function is computable (in the intuitive sense) iff it is partial recursive. We can thus claim with reasonable confidence that we have characterised the computable functions.

We turn back to more technical matters now. A syntactic expression involving 0, S, P^n_k, Cn, Pr, and Mn that is formed according to the rules of Definitions 1.1, 1.2, 1.3, 1.6 is called *operator term*. We write $\mathcal{O}$ for the set of all operator terms and, for $k \in \mathbb{N}$, $\mathcal{O}_k$ for the set of all operator terms defining a k-ary function. For example, $\mathrm{Pr}[\mathrm{P}^1_1, \mathrm{Cn}[S, \mathrm{P}^3_3]] \in \mathcal{O}_2$. The primitive recursive (partial recursive) functions are closed under definition by cases:

Lemma 1.2. *If $g, f_0, \ldots, f_n : \mathbb{N}^k \hookrightarrow \mathbb{N}$ are primitive recursive (partial recursive), then so is $h : \mathbb{N}^k \hookrightarrow \mathbb{N}$ defined by*

$$h(\overline{x}) = \begin{cases} f_0(\overline{x}) & \text{if } g(\overline{x}) = 0 \\ f_1(\overline{x}) & \text{if } g(\overline{x}) = 1 \\ \vdots & \\ f_{n-1}(\overline{x}) & \text{if } g(\overline{x}) = n-1 \\ f_n(\overline{x}) & \text{if } g(\overline{x}) \geq n \\ \text{undefined} & \text{if } g(\overline{x}) \uparrow \end{cases}$$

Proof. We have $h(\overline{x}) = \chi_=(g(\overline{x}), 0) \cdot f_0(\overline{x}) + \cdots + \chi_=(g(\overline{x}, n-1) \cdot f_{n-1}(\overline{x}) + \chi_\geq(g(\overline{x}, n), f_n(\overline{x}))$. □

Example 1.3. $\min, \max : \mathbb{N}^2 \to \mathbb{N}$ are primitive recursive, since

$$\min(x, y) = \begin{cases} x_1 & \text{if } x_1 \leq x_2 \\ x_2 & \text{otherwise} \end{cases}, \text{ and}$$

$$\max(x, y) = \begin{cases} x_1 & \text{if } x_1 \geq x_2 \\ x_2 & \text{otherwise} \end{cases}.$$

1.2 Undecidability

Definition 1.8. A relation $R \subseteq \mathbb{N}^k$ is called *decidable* if $\chi_R : \mathbb{N}^k \to \{0, 1\}$ is recursive.

Theorem 1.1. *There are undecidable sets.*

Proof. Every operator term is a finite string of symbols which are taken from a countable set. Therefore, there are only countably many operator terms, hence there are only countably many partial recursive functions, and thus, only countably many decidable relations. On the other hand, there are uncountably many $A \subseteq \mathbb{N}$. □