Computability theory

These notes cover the second part of PMATH 432/632 taught in the Winter semester of 2020

Dino Rossegger

Compiled: July 27, 2020

1. Introduction to computability theory

Our goal is to develop machinery that captures the intuitive notion of computability of a function $f : \mathbb{N} \to \mathbb{N}$. Intuitively a function is said to be computable if there is an *algorithm* for it: a fixed finite sequence of atomic instructions so that given a natural number x after applying the actions defined by my instructions to it, I obtain $f(x)$. The instructions do not depend on x. Of course to develop a theory of computability we have to formalize the notion of an algorithm. There are several ways to do this. Historically the three first were the (primitive) recursive functions (Gödel, Kleene), λ-calculus (Church), and Turing machines (Turing), which all emerged independently roughly around the same time. We will focus on Turing's model of computation, Turing machines, introduced in his paper "On computable numbers, with an application to the Entscheidungsproblem" in 1937.

Since then several other definitions such as Register machines have emerged, each with its own advantages and disadvantages. However, one can show that all models of computations are equivalent in the sense that a function is computable in one model if and only if it is computable in all of them.

A *Turing machine* T consists of the following components:

1. A tape, infinite in both directions, whose cells contain either the symbol 1 or the symbol 0 "blank"
2. a reading head which can move to the right or left on the tape, read the content of its current cell and replace it by 1 or 0.

At any given time the reading head is in some state q and follows the *Turing program* P.

The Turing program will be a set of instructions for the reading head, each instruction being a quadruple:

$$Q = q_i S A q_j \text{ with } S \in \{0,1\},\ A \in \{0,1,L,R\}$$

The quadruple Q reads as: "If T is in the state q_i reading tape symbol S, then perform action A and pass into the new internal state q_j. If T is in the state q_i, reading S, then we say that Q is applicable.

Definition 1.1. A set X of quadruples is *consistent* if

$$q_i S A q_j,\ q_i S A' q_k \in X \Rightarrow A = A' \text{ and } q_k = q_j.$$

A *Turing program* is a finite consistent set of quadruples.

Notice how the behaviour of a Turing machine only depends on its program. We often say "give a Turing machine for f" or there is a Turing machine when we mean that there is a Turing program for a Turing machine.

How can we compute something with a Turing machine, say we want to compute a function $f : \mathbb{N}^n \to \mathbb{N}$. We will stick with the following conventions.

To input $(n_0, \dots, n_n)$ we put $n_0 + 1$ consecutives 1 on the otherwise empty tape, then put a blank, then put $n_1 + 1$ consecutives 1 followed by a blank and so on. We then place the reading head on the leftmost 1.

If the computation of a Turing machine T *halts*, output the number $\varphi_T(n_0, \dots, n_n)$ of 1's left on the tape.

Definition 1.2. A Turing machine *halts* on input n if at some point during its computation there is no applicable quadruple in its program.

Definition 1.3. A function $f : \mathbb{N}^n \to \mathbb{N}$ is *Turing computable* if $f = \varphi_T$ for some Turing machine T.

Example 1.1. *The successor function $n \mapsto n + 1$ is Turing computable.*

Proof. Let $T = \{\}$. □

Example 1.2. *The* zero function $\mathbf{0} : n \to 0$ *is Turing computable.*

Proof. P consists of the following quadruples:

1. $q_0 1 0 q_1$ ...delete the one at the current position, then go to q_1
2. $q_1 0 R q_0$ ...move the reading head to the right and get back to q_0 to delete more 1's.

Then $\varphi_T(n) = 0$ for all $n \in \mathbb{N}$. □

Remark. Notice that the Turing computable functions are not necessarily total, indeed they might not be defined at some values, as a Turing machine may not halt at some values.

We write $\varphi_T(n) \downarrow$ to say that the function computed by the Turing machine T is defined on n. Notice that it is defined only if T halts on input n. Also, given a partial function f we let $dom(f)$ be the set of values where f is defined and $rg(f)$ the range of f.

Remark. A function $f : \mathbb{N}^n \to \mathbb{N}$ is partial if it is not defined on some elements of $\mathbb{N}^n$. Otherwise it is total.

Thesis 1 (Turing's thesis)**.** A function f is Turing computable if and only if it is intuitively computable.

Church gave a similar thesis but for partial recursive functions. One can show that the partial recursive functions are exactly the Turing computable functions. Therefore this thesis is often also called the Church-Turing thesis.

Supported by this thesis, we can show that something is computable by arguing informally that an algorithm in whatever system exists and do not rely on the heavy formalism involved in Turing machines. We therefore refer to the Turing computable functions as computable functions. We will state in case that a function is total.

Remark. Computable functions are often called recursive functions or partial recursive functions. This has roots in the fact that they are precisely those functions which are partial recursive. However, by the Church-Turing thesis, the name computable functions is warranted, as the "Turing computable" or "partial recursive" functions are precisely the computable functions.

2. Universal Turing machine

Using a Gödel numbering we can obtain a computable list of all Turing machines. We define the function gn, first on tape symbols, action symbols and state symbols. Let $gn(L) = 2, gn(R) = 3, gn(0) = 5, gn(1) = 7, qn(q_i) = p_{4+i}$ where p_i is the i^{th} prime number. For a quadruple $Q = q_i S A q_j$ let

$$gn(Q) = 2^{gn(q_i)} 3^{gn(S)} 5^{gn(A)} 7^{gn(q_j)}$$

and if $P = \{Q_0, Q_1, \ldots, Q_k\}$ is a Turing program, let

$$gn(P) = 2^{gn(Q_0)} \ldots p_{k+1}^{gn(Q_k)}.$$

By unique prime factorization gn has a computable inverse. Note that not for every natural number, this inverse is indeed a Turing machine.

Definition 2.1. 1. The e^{th} Turing machine P_e is defined by

$$P_e = \begin{cases} P & \text{if } gn^{-1}(e) \downarrow = \text{some Truing program } P \\ \emptyset & \text{otherwise.} \end{cases}$$

We say that P_e has index e.

2. Likewise we let $\varphi_e^{(k)}$ be the partial computable function in k arguments computed by P_e. If the arity is clear form context we drop the superscript.

Theorem 2.1 (Enumeration Theorem). *There is a partial computable function φ such that for each e and x $\varphi(e,x) = \varphi_e(x)$.*

Proof. We need to describe a Turing machine T computing φ. The computation works as follows:

1. Compute P_e.
2. Put x on the tape and run the computation P_e.
3. If P_e halts, halt as well.

$\square$

Theorem 2.2 (s-m-n Theorem). *If $f(x,y)$ is partial computable, then there exists a total computable function g such that $f(x,y) = \varphi_{g(x)}(y)$.*

Proof. If $f(x,y)$ is partial computable, then there is a Turing machine P_e such that $f(x,y) = \varphi_e^{(2)}(x,y)$. Fix x, and let $g(x)$ be Gödel number of the program doing the following:

1. Put $x+1$ consecutive 1's to the left of the last blank before a 1 on the tape.
2. Position the cursor on the left most 1.
3. Run P_e.

Clearly $P_{g(x)}$ with one argument behaves like P_e with 2 arguments where the first argument is x. $\square$

The s-m-n theorem can be stated in a more general context.

Exercise 2.1. Let $m,n \geq 1$. Show that there exists a total computable function S_n^m such that for all $x, y_1, \dots y_m, z_1, \dots, z_n$

$$\varphi^{(n)}_{S_n^m(x,y_1,\dots,y_m)}(z_1,\dots,z_n) = \varphi_x^{(n+m)}(y_1,\dots,y_m,z_1,\dots,z_m).$$

The main intellectual contribution of Turing's paper was not the invention of the Turing machine but rather that he showed that there is a universal Turing machine that can perform any computational task.

Theorem 2.3 (Turing). *There exists a Turing machine U – the Universal Turing machine – which if given input (e,x) simulates the e^{th} Turing machine with input x. That is $\varphi_U^{(2)}(e,x) = \varphi_e(x)$.*

Proof. The Turing machine computing the function φ from Theorem 2.1 is such a Turing machine. $\square$

Theorem 2.4 (Kleene's fixed point theorem, also recursion theorem)**.** *If f is a total computable function, then there exists a $k \in \mathbb{N}$ for which*

$$\varphi_{f(k)} = \varphi_k.$$

k is called a fixed point of f.

Proof. Define the function d by

$$\varphi_{d(u)}(z) = \begin{cases} \varphi_{\varphi_u(u)}(z) & \varphi_u(u)\downarrow \\ \uparrow & \text{otherwise.} \end{cases}$$

Note that d can be taken total and computable by the s-m-n theorem. As f is computable we have that $f \circ d$ has an index v such that $\varphi_v = f \circ d$. As both f and d are total so is $f \circ d$ and thus $\varphi_v(v)\downarrow$ and thus $\varphi_{d(v)} = \varphi_{\varphi_v(v)}$. Now

$$\varphi_{d(v)} = \varphi_{\varphi_v(v)} = \varphi_{f(d(v))}$$

and thus $d(v)$ is a fixed point of f. □

Corollary 2.5. *There exists $e \in \mathbb{N}$ such that $rg(\varphi_e) = e$.*

Proof. Consider the function $f(x, y) = x$. It is a computable function so it has an index i. By the $s - m - n$ theorem there is a total computable function g such that $\varphi_{g(x)}(y) = x$. Now, by the recursion theorem this function has a fixed point v, i.e., $\varphi_{g(v)} = \varphi_v$ and by definition $rg(\varphi_{g(v)}) = rg(\varphi_v) = v$. □

2.1. Computable approximations

We will shortly see that there is no algorithm that tests whether $\varphi_e(x)\downarrow$ or $\varphi_e(x) = y$.

Definition 2.2. We write

1. $\varphi_{e,s}(x) = y$ iff $x, y, e < s$, $\varphi_e(x) = y$ and the computation of P_e on input x halts in s steps,

2. $\varphi_{e,s}(x)\downarrow$ if $\varphi_{e,s}(x) = y$ for some y and otherwise we write $\varphi_{e,s}(x)\uparrow$.

Theorem 2.6. *All the relations in Definition 2.2 are computable (Computability of relations is defined in 2.2).*

Theorem 2.7 (Unbounded search theorem)**.** *If $\theta(x, y)$ is a partial computable function, and*

$$\psi(x) = \mu y[\theta(x, y)\downarrow= 1 \quad \& \quad \forall z < y \theta(x, z)\downarrow\neq 1],$$

then $\psi(x) = y$ is partial computable.

Proof. Fix x. Now, as θ is computable, there is an e such that $\theta = \varphi_e$. Now for some fixed s compute $\varphi_{e,s}(x,y)$ for all $y < s$. If you find y such that $\varphi_{e,s}(x,y) = 1$ and $\varphi_{e,s}(x,z) \downarrow\neq 1$ for all $z \leq y$, then halt and set $\psi(x) = y$, otherwise repeat this procedure with $s+1$ instead of s.

By the Church-Turing thesis it is computable and $\psi(x) = y$ if y is the least satisfying

$$\theta(x,y) \downarrow= 1 \quad \& \quad \forall z < y \theta(x,z) \downarrow\neq 1.$$

Thus ψ is as required. $\square$

In computability theory μy is used to denote the "least y such that". This is inspired by the recursive functions as they have the minimization operator μ as one of their basic components.

2.2. Computability of sets

Definition 2.3. A set S is computable if its characteristic function

$$\chi_S(x) = \begin{cases} 1 & x \in S \\ 0 & x \notin S \end{cases}$$

is computable.

We could give analogous definitions for relations $R \subseteq \mathbb{N}^n$. However, there is no need for this since we have a computable bijection $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$. One well known example is the *Cantor pairing function* defined by

$$\langle k_1, k_2 \rangle = \frac{1}{2}(k_1 + k_2)(k_1 + k_2 + 1) + k_2.$$

We can inductively generalize this definition to obtain pairing functions for arbitratry n, e.g. for $\mathbb{N}^3 \to \mathbb{N}$ we let $\langle x, y, z \rangle = \langle \langle x, y \rangle, z \rangle$. We thus can make the following definition.

Definition 2.4. A relation $R \subseteq \mathbb{N}^n$ is computable if $\{\langle x_1, \ldots, x_n \rangle : (x_1, \ldots, x_n) \in \mathbb{N}^n\}$.

Proposition 2.8. *If X is a computable set, then so is $\bar{X} = \mathbb{N} - X$.*

Note that there are only countably many algorithms but uncountably many subsets of the natural numbers. Therefore there must be non-computable sets.

Exercise 2.2. Show that if X is a computable set and Y is defined by $x \in Y \Leftrightarrow \exists y < x \langle x, y \rangle \in X$, then Y is computable.

Exercise 2.3. Show that any finite or cofinite set is computable.

3. C.e. sets

Definition 3.1. A set S is computably enumerable, short c.e., if it is empty or the range of a partial computable function.

Theorem 3.1. *A set S is c.e. iff it is empty or the range of a total computable function.*

Proof. The direction from right to left is immediate from the definition. Now, assume S is c.e. and non-empty, then it is the range of a partial computable function, say f given by index e. Let $y \in S$ and define a new function g by

$$g(\langle x, s\rangle) = \begin{cases} \varphi_e(x) & \varphi_{e,s}(x)\downarrow \\ y & \textit{otherwise.} \end{cases}$$

By the Church-Turing thesis g is computable and by definition it is total. Clearly the range of g is S. □

Proposition 3.2. *An infinite set S is computable iff it is the range of a strictly monotonic computable function.*

Proof. ($\Rightarrow$). Say S is computable and define a function f by $f(0) = \mu y[y \in S]$ and $f(x) = \mu y[y \in S \& y > f(x-1)]$.

($\Leftarrow$). Say S is the range of an increasing computable function f. Then $x \in S$ iff $\exists y \leq x(f(y) = x)$. □

Theorem 3.3. *A is computable iff A and $\bar{A} = \mathbb{N} - A$ are c.e.*

Proof. Say $\mathcal{A}$ is computable, then so are the functions

$$f(x) = \begin{cases} x & x \in A \\ \uparrow & \text{otherwise} \end{cases} \quad \text{and} \quad g(x) = \begin{cases} x & x \notin A \\ \uparrow & \text{otherwise.} \end{cases}$$

Clearly $rg(f) = A$ and $rg(g) = \bar{A}$. Thus both A and $\bar{A}$ are c.e.. On the other hand say A and $\bar{A}$ are both c.e.. To check whether $x \in A$ start enumerating A and $\bar{A}$ in parallel. At some point x must be enumerated into A or into $\bar{A}$. Then output 1 if x gets enumerated into A and 0 if x gets enumerated into $\bar{A}$. □

Definition 3.2. Let $W_e = dom(\varphi_e)$. We call W_e the *halting set* for φ_e. Similarly to computable functions we define the set $W_{e,s}$ for all s as

$$W_{e,s} = \{x : \varphi_{e,s}(x)\downarrow\}.$$

It is easy to check that $W_{e,s}$ is a computable set.

Theorem 3.4 (The normal form theorem)**.** *Let A be a set. TFAE*

1. *A is c.e.,*
2. *the semicharacteristic function of A,*

$$S_A(x) = \begin{cases} 1 & x \in A \\ \uparrow x \notin A \end{cases}$$

   is computable,

3. *A is Σ^0_1, i.e., there is a computable relation R such that $x \in A$ iff $\exists y R(x,y)$,*
4. *$A = W_e$ for some $e \in \mathbb{N}$.*

Proof. (1) $\Rightarrow$ (3). If $A = \emptyset$ then $x \in A \Leftrightarrow \exists s x = x+1$. Otherwise $A = range(f)$ where f is computable. Then $x \in A \Leftrightarrow \exists s f(s) = x$. (3) $\Rightarrow$ (2). Say $x \in A \Leftrightarrow \exists s R(x,s)$. Then $S_A(x) = 1 \Leftrightarrow \exists s R(x,s)$ which is clearly partial computable. (2) $\Rightarrow$ (4). Assume S_A is computable, then $S_A = \varphi_e$ for some e and by definition $W_e = A$. (4) $\Rightarrow$ (1). Assume $A = W_e \neq \emptyset$ and let $p \in A$. Define

$$f(\langle x, s\rangle) = \begin{cases} x & \varphi_{e,s}(x)\downarrow \\ p & \text{otherwise} \end{cases}$$

Both f and $\langle \bullet \rangle$ are computable and we have that

$$x \in A \Leftrightarrow \varphi_e(x)\downarrow \Leftrightarrow \exists s \varphi_{e,s}(x)\downarrow \Leftrightarrow f(x,s) = x \text{ for some } s$$

and thus $A = range(f)$. □

Exercise 3.1. Let $A \subseteq \mathbb{N}$ and f be a total computable function. Show that if

$$x \in A \Leftrightarrow \exists y < f(x) \langle x, y\rangle \in S$$

for S a computable set, then A is computable.

Definition 3.3. The Kleene set K is defined as follows.

$$K = \{x : \varphi_x(x)\downarrow\}$$

Theorem 3.5. *There exists a c.e. non-computable set.*

Proof. The set K is clearly c.e. as

$$x \in K \Leftrightarrow x \in W_x \Leftrightarrow \exists s x \in W_{x,s}$$

and $W_{x,s}$ is a computable relation.

Assume K was computable, then $\bar{K}$ is c.e., i.e. $\bar{K} = W_e$ for some e. Then

$$x \in W_e \Leftrightarrow x \in \bar{K} \Leftrightarrow x \notin K \Leftrightarrow x \notin W_x.$$

Let $x = e$ to derive a contradiction. □

4. Many one reducibility and the Entscheidungsproblem

Definition 4.1. We say that A is many-one or m-reducible to B, $A \leq_m B$ if there is a total computable function f such that for all x

$$x \in A \Leftrightarrow f(x) \in B.$$

We write $A \equiv_m B$ if $A \leq_m B$ and $B \leq_m A$. We call

$$deg_m(A) = \{B : B \equiv_m A\},$$

the m-degree of A.

Exercise 4.1. Show that if X is c.e. and $Y \leq_m X$, then Y is c.e.

Exercise 4.2. Show that if X is c.e. and non-computable and $X \leq_m Y$, then Y is non-computable.

Proposition 4.1. *Let X be any c.e. set, then $X \leq_m K$.*

Proof. If X is c.e., then $S_X = \varphi_e$ for some e. Define a new function f as

$$f(x, y) = \varphi_e(x)$$

Now clearly, $f(x, y)$ is computable, i.e., $f = \varphi_i$ for some i. Then, by the s-m-n theorem there is a computable function g such that

$$\varphi_i(x, y) = \varphi_{g(x)}(y).$$

Then

$$x \in X \Leftrightarrow \varphi_e(x) \downarrow \Leftrightarrow \varphi_{g(x)}(g(x)) \downarrow \Leftrightarrow g(x) \in K.$$

□

Proposition 4.2. *Let $K_\emptyset = \{x : \varphi_x(0) \downarrow\}$. Then $K_\emptyset \equiv_m K$.*

Proof. To show that $K_\emptyset \leq_m K$ it is sufficient to show that it is c.e. $K_\emptyset$ is Σ^0_1 as $x \in K_\emptyset \Leftrightarrow \exists s \varphi_{x,s}(0) \downarrow$ and thus it is c.e. Define a function f by $f(x, 0) = \varphi_x(x)$ and $f(x, y) = 0$ otherwise. f is clearly partial computable and thus there is e such that $f = \varphi_e$. By the s-m-n theorem we have that there is computable g such that

$$\varphi_e(x, y) = \varphi_{g(x)}(y).$$

Then

$$x \in K \Leftrightarrow \varphi_x(x) \downarrow \Leftrightarrow \varphi_{g(x)}(0) \downarrow \Leftrightarrow g(x) \in K_\emptyset$$

□

Definition 4.2. Let $\Gamma \subseteq \mathcal{P}(\mathbb{N})$. Then a set $X \in \Gamma$ is Γ (m-)complete if for every $Y \in \Gamma$ $Y \leq_m X$.

Proposition 4.3. *K is Σ^0_1 complete.*

Proof. K is c.e. and thus also Σ^0_1. By Proposition 4.1 K is Σ^0_1 complete. □

4.1. The Entscheidungsproblem

The Entscheidungsproblem (Entscheidung german for decision) was a problem of Hilbert's posed in 1928 that asked whether there is an algorithm that determines whether $\models \varphi$ for any first order formula φ. The Entscheidungsproblem was solved in the negative first by Church and one year later independently by Turing.

Recall that S^∞ is the symbol set having infinitely many constants and infinitely many relation and function symbols for any arity. Fix a Gödel-numbering of all S^∞ formulas and let ψ_i be the i^{th} formula in this numbering. We show the following.

Theorem 4.4. $K \leq_m \{i :\models \psi_i\}$.

As $\bar{K}$ is not computable we immediately obtain:

Corollary 4.5. *There is no algorithm deciding whether a given first order formula is true.*

To prove Theorem 4.4 we need to give a computable function f such that P_e does not halt on input e iff $f(e)$ is unsatisfiable. We obtain f_e by coding Turing machines into sentences. We now work towards describing the sentence $f(e)$ given e.

First notice that at every given time the state of a Turing machine is described by (a) the position of its head, (b) the content of the tape, (c) the state it is in. We thus will define a two dimensional grid infinite to the left and the right so that the y-axis corresponds to the states of a Turing machine during its computation and the x-axis corresponds to the tape. We will have a constant 0 which is the position of our head at the start of the program, and will split the grid into a right and left part. To obtain the grid we need the relation symbol $B/1$ with the idea that $B/1$ holds of the bottom row, the constant symbol 0, and the function symbols $l/1$, $r/1$, $u/1$ allowing us to traverse the grid left, right and up respectively. We also need relation symbols $L/1, R/1, O/1$ saying that an element is in the left or right part of the grid or on the origin (the column of 0).

One possible axiomatization of the grid is as follows:

$$B0 \tag{1}$$

$$\forall x(Bx \leftrightarrow \exists y(By \wedge ly \equiv x)) \tag{2}$$

$$\forall x(Bx \leftrightarrow \exists y(By \wedge ry \equiv x)) \tag{3}$$

$$\forall x\ \neg Bux \tag{4}$$

$$\forall x(urx \equiv rux \wedge lux \equiv ulx \wedge lrx \equiv x \wedge rlx \equiv x) \tag{5}$$

$$\forall x(\neg lx \equiv x \wedge \neg rx \equiv x \wedge \neg ux \equiv x) \tag{6}$$

$$\forall x \forall y\ (nx \equiv ny \to x \equiv y) \tag{7}$$

$$\forall x(Rx \vee Lx \vee Ox) \wedge \neg(Rx \wedge Lx) \wedge \neg(Lx \wedge Ox) \wedge \neg(Rx \wedge Ox) \tag{8}$$

$$O0 \tag{9}$$

$$\forall x(\neg Lx \vee \neg Rx) \tag{10}$$

$$\forall x((x \equiv l0 \to Lx) \wedge (x \equiv r0 \to Rx)) \tag{11}$$

$$\forall x((Lx \to Llx) \wedge (Rx \to Rrx)) \tag{12}$$

Now, fix P_e and assume that P_e has states $q_0, \dots q_k$. We will work with the relation symbols $S_0/1$, $S_1/1$ for the tape symbols, $Q_0/1, \dots, Q_k/1$ for the states, and $H/1$ for the head. The following sentences axiomatize the Turing machine. We write $r^i x$ for the term $\underbrace{r \dots r}_{\text{i times}} x$.

$$\forall x((S_0 x \vee S_1 x) \wedge \neg(S_0 x \wedge S_1 x)) \tag{13}$$

$$\forall x Bx \to (x = 0 \leftrightarrow Hx) \wedge Q_0 0 \wedge \bigwedge_{0<i\leq k} \neg Q_k 0 \wedge (\bigwedge_{i \leq e} x = r^i 0 \leftrightarrow S_1 x) \tag{14}$$

$$\forall x \neg Hx \to (S_1 ux \leftrightarrow S_1 x) \tag{15}$$

$$\forall x (Hx \leftrightarrow \bigvee_{i \leq k} Q_i x) \tag{16}$$

Now let $q_i S A q_j \in P_e$. Assume for demonstration purposes that $S = 1$ and $A = L$, then we add the sentence

$$\forall x((Hx \wedge S_1 x \wedge Q_i x) \to (Hulx \wedge Q_j ulx)). \tag{17}$$

We add similar rules for other types of quadruples.

We still need to add statements saying that if in some row we see a given state, then this states needs to be specified by the program. We add for all $0 < i \leq k$ and all $j \in 0, 1$ the following sentence:

$$\begin{aligned}&\forall x \Bigg((Hx \wedge Q_i x \wedge S_j x) \to \Bigg(x = 0 \\ &\vee \exists y \Bigg(\Bigg(\Bigg(uly = x \wedge \bigvee_{q_l S_m L q_i \in P_e} (Hy \wedge S_m y \wedge Q_l y) \Bigg) \vee \Bigg(uy = x \wedge \bigvee_{q_l S_m j q_i \in P_e} (Hy \wedge S_m y \wedge Q_l y) \Bigg) \\ &\vee \Bigg(ury = x \wedge \bigvee_{q_l S_m R q_i \in P_e} (Hy \wedge S_m y \wedge Q_l y) \Bigg) \Bigg) \Bigg) \Bigg) \Bigg)\end{aligned} \tag{18}$$

At last we need to add a sentence that says that P_e halts. Recall that P_e halts on input e if at any point during its computation it is in a state such that no rule is applicable. Now, looking at P_e we can extract the "states" at which no rules are applicable,i.e., pairs q_iS where $S \in \{0, 1\}$. Let H be the set of such states, we want to find a sentence stating that $P_e(e)$ does not halt. If $H = \emptyset$ let $\psi_H = \forall x\ x \equiv x$, otherwise

$$\psi_H = \forall x (Hx \to \bigwedge_{q_i j \in H} \neg(Q_i x \wedge S_j x)). \tag{19}$$

Lemma 4.6. *$Sat(1)-(19)$ iff $P_e(e)$ does not halt.*

Proof sketch. ($\Leftarrow$). Say $P_e(e)$ does not halt. We can build a model $\mathcal{A}$ of the sentences (1) to (19) as follows. The universe of $\mathcal{A}$ is $\mathbb{N}$ and we view $\mathbb{N}$ under the pairing function $\langle x, y\rangle$. Now we define the relations for $\langle x, -\rangle$ canonically by looking at $P_e(e)$ after x steps of its computation.

($\Rightarrow$). If $Sat(1)-(19)$ then it will have many models. First of all using compactness one can show that it has countable non-standard models and using Löwenheim-Skolem it will have models in any cardinality. However axiom $(1)-(12)$ implies that it has a standard part which behaves like an upper half plane. Given a countable model $\mathcal{A}$ we can argue that it models the computation of $P_e(e)$ as follows:

At stage 0, $P_e(e)$ is in state q_0 with the head on the first of e consecutive 1's and all other cells of the tape are blank. As $\mathcal{A} \models (14)$ the first line of the upper half plane in $\mathcal{A}$ models this behaviour.

By $\mathcal{A} \models (13), (15)-(18)$ we have that every row of $\mathcal{A}$ models a computational state of the Turing machine and by $\mathcal{A} \models (17)$ and by induction $\mathcal{A}$ models the computation of P_e in its standard rows. Now, as $\mathcal{A} \models (19)$ there is no row modeling a halting state (a state in which no rule is applicable). So in particular, $P_e(e)$ can never reach a halting state, as there is no halting state modelled in the standard rows. □

Exercise 4.3. Use induction to give a full formal proof of Lemma 4.6.

Proof. Proof of Theorem 4.4. We have by Lemma 4.6 that $P_e(e)$ does not halt iff $Sat(1)-(19)$. In particular

$$e \in \bar{K} \Leftrightarrow Sat(1)-(19).$$

Let θ_e be the negation of the conjunction of sentences in (1) to (19). As the Turing program P_e is finite, this is a first order sentence and as $\models \theta_e$ iff $Unsat\neg\theta_e$ we have $e \in K \Leftrightarrow \models \theta_e$. Fix a Gödel numbering gn of all S^∞ formulas and let $f : e \to gn(\theta_e)$ (Similar to the case for Turing machines we can take gn to be a bijection by saying that if a number is not a Gödel number of a first order formula, then let $gn(x)$ be the code of $\forall x x \equiv x$). Looking at the sentences (1)-(19) one sees that in order to compute f it is sufficient to look at the Turing program coded by e to obtain θ_e and then take $gn(\theta_e)$ to obtain f. Thus, by the Church Turing thesis f is computable. Let V be the set of valid S^∞ sentences, i.e., $V = \{x :\models gn^{-1}(x)\}$. Therefore $e \in K \Leftrightarrow f(e) \in V$, and hence $K \leq_m V$. By Exercise 4.2 V can not be computable and thus there can not be an algorithm deciding whether a first order formula is valid nor not. □

Note that while the Entscheidungsproblem was stated for arbitrary formulas, our proof can be easily adapted to show something slightly stronger, namely that there is no algorithm deciding first order sentences.

One can show that if the axioms of a proof system are computable, then the set of statements provable from it is c.e. We will only describe this informally here but will go into more detail in the next section. Recall the definition of our proof calculus, it was an inductive definition, having as its elements sequents and allowing us to derive new sequents from already derived sequents by the application of finitely many rules. So, if the set of codes for starting sequents (axioms) is computable, to check whether $x \in V$ we have to ask, does there exist a finite sequence of sequents, one derived from the ones before by application of rules such that the last sequent ends with φ_x. Alone from reading this description we may see that this is Σ^0_1 description for the set of valid sentences in a given system. We thus obtain as a corollary.

Corollary 4.7. *The set of true S_∞ formulas is Σ^0_1 complete.*

Exercise 4.4. Formalize the above considerations.

We will take a quick detour to give another example of Σ^0_1 complete set.

4.2. Hilbert's Tenth problem

So far we have seen incomputable sets which arise out of our definitions or out of logic. A computer scientist or logician might consider these sets natural, but for a common mathematician they must seem quite unnatural and uninteresting. This might lead the working mathematician to the conclusion that logic and computability are not of importance to his work. We now turn to an example of a set which is not computable and which most mathematician would encounter naturally.

The problem we are interested is related to polynomial equations: Let f be a polynomial equation in one or more variables with integer coefficients (such an equation is usually called *Diophantine*, does f have an integer solution?

When Hilbert posed his famous 23 problems at the International Congress of Mathematicians, the following was number 10 on his list:

Question 4.1 (Hilbert's tenth problem)**.** Find an algorithm that decides whether a given Diophantine equation has an integer solutions.

It took 70 years and over 30 years of continuous work by several mathematicians to solve this question in the negative: It is impossible to find such an algorithm.

Theorem 4.8 (Matiyasevitch, Davis ,Putnam, Robinson)**.** *There is no algorithm deciding whether a given Diophantine equation has an integer solution.*

Let us state this more formally. Fix a Gödel numbering for Diophantine equations. Then the set

$$D = \{gn(p) : \exists y_1, \dots, y_n \; p(y_1, \dots, y_n) = 0\}$$

is not computable. Note that this set is naturally c.e., we can evaluate a polynomial at given values, so we can search for solutions. However one can show that this set Σ^0_1-complete.

The idea of the proof is the following. We say that a set A is *Diophantine* if $A = \{x : \exists y_1, \dots y_n p_A(x, y_1, \dots, y_n) = 0\}$ where p_A is some polynomial with integer coefficients. The idea for a proof of Theorem 4.8 is due to Davis: "Show that any c.e. set is diophantine." Why would this give a proof? Consider the set K. If K was Diophantine there would be a polynomial p_K such that

$$K = \{x : \exists y_1, \dots, y_n p_K(x, y_1, \dots, y_n) = 0\}.$$

Now, consider the function $f : n \to gn(p_K \frac{n}{x} = 0)$. It is a computable function such that $x \in K$ iff $f(x) \in D$, so $K \leq_m D$. Julia Robinson verified for large classes of c.e. sets that they were indeed Diophantine and in 1960 Putnam and Robinson were able to show that every c.e. set is Diophantine iff there is one exponentially increasing set (i.e., if we enumerate it in order, then the sequence is exponential) that is Diophantine. In 1970 the until then unknown Russian mathematician Yuri Matiyasevitch showed that the Fibonacci sequence is Diophantine, thus solving Hilbert's tenth problem in the negative.

5. Gödel's incompleteness theorems

Remark. On the way to prove the incompleteness theorems we have to prove a couple of technical lemmas. We will not give the proofs of these theorems in full detail and instead only provide sketches. We hope that the details provided are sufficient for the interested reader to fill them out on his own.

5.1. Peano arithmetic

The Peano axioms were formulated by Guiseppe Peano at the end of the 19th century to formalize arithmetic. They are widely considered to capture arithmetic on the natural numbers and thus to be a natural axiomatization for the structure $\mathbb{N}$ in S_{ar}. The axioms are:

$$\begin{aligned} \forall x \neg x + 1 \equiv 0 \qquad & \forall x \forall y (x + 1 \equiv y + 1 \to x \equiv y) \\ \forall x x + 0 \equiv x \qquad & \forall x \forall y x + (y + 1) \equiv (x + y) + 1 \\ \forall x x \cdot 0 \equiv 0 \qquad & \forall x \forall y x \cdot (y + 1) \equiv x \cdot y + x \end{aligned}$$

And the first order induction axiom for every formula $\varphi \in L^{S_{ar}}$ and all $x_1, \dots, x_n, y$ such that $free(\varphi) \subseteq \{x_1, \dots, x_n, y\}$:

$$\forall x_1, \dots x_n \left((\varphi \frac{0}{y} \wedge \forall y (\varphi \to \varphi \frac{y+1}{y})) \to \forall y \varphi \right).$$

We write Φ_{PA} or just PA for the set of axioms, and also call sets of sentences theories. (We deviate here from Ebbinghaus et al. who assume a theory to be closed under consequence and consistent.)

As usual let $\mathbf{m}$ stand for $\underbrace{1 + \cdots + 1}_{m \text{ times}}$ and $\bar{x}$ for a tuple $(x_1, \dots, x_n)$ for some n.

Remark. We will use the notation as suggested in Ebbinghaus p37 to deal with free variables. In particular we will write $\varphi(x, y, z)$ for a formula φ with free variables x, y, z and $\varphi(a, b, c)$ for the formula

$$\varphi\frac{a}{x}\frac{b}{y}\frac{c}{z}$$

Definition 5.1. A relation $R \subseteq \mathbb{N}^k$ is *representable* in a set of S_{ar} sentences Φ if there is φ such that

$$\bar{m} \in R \Rightarrow \Phi \vdash \varphi(\bar{\mathbf{m}}), \text{ and}$$
$$\bar{m} \notin R \Rightarrow \Phi \vdash \neg\varphi(\bar{\mathbf{m}}).$$

A function $f : \mathbb{N}^k \to \mathbb{N}$ is *representable* in Φ if the graph of f $Graph_f = \{\langle \bar{x}, y\rangle : f(\bar{x}) = y\}$ is representable.

Which functions are representable in $Th(\mathbb{N})$ or PA? We will argue that all computable functions are representable in the standard model of arithmetic, i.e., in $Th(\mathbb{N})$. With some extra work the avid student could prove that they are indeed representable in PA. We will state the theorem for this as we will need them for our proof of the incompleteness theorem but wont give proofs. We refer the reader to [Men09] for details.

When one tries to represent functions in $\mathbb{N}$, one quickly runs into obstacles. For instance consider exponentiation: x^y. Then we would usually define exponentiation inductively by

$$x^0 = 1 \quad x^{y+1} = x^y \cdot x.$$

So in order to say that $x^y = z$ we would have to formulate that there is a sequence of numbers of length y starting at 1, progressing as in the definition of exponentiation and ending at z. But in $Th(\mathbb{N})$ or PA we can only talk about numbers and not about sequences of numbers. So how do we represent sequences?

Gödel solved this problem in order to prove the incompleteness theorem by using the Chinese remainder theorem, using it to define his β function which codes sequences of natural numbers.

Lemma 5.1. *Let $\beta(x, y, z) = rm(x/(1+(z+1)\cdot y))$ (rm is the remainder function). Then β is computable and representable in the standard model of arithmetic (in PA).*

Remark. We give definitions using $<$. By the assignment it is clear that $<$ is definable in S_{ar}.

Proof. By the Church Turing thesis division with remainder is computable, so β is computable. To see that it is representable in the standard model consider the formula

$$\varphi_\beta(x, y, z, u) = \exists w\,(x \equiv (1 + (z + 1) \cdot y) \cdot w + u \wedge u < (1 + (z + 1)) \cdot y)$$

We can argue semantically that this indeed represents β in $\mathbb{N}$. $\square$

Notice that while the definition of φ_β as given is Σ_1, the existential quantifier is bounded. Indeed it is sufficient to look at w which are smaller than x to see whether φ_β holds of given x, y, z, u. This is important because by Exercise 3.1 we obtain that evaluating φ_β in the standard model is computable.

Lemma 5.2 (β-function lemma)**.** *For any sequence of natural numbers $n_0, \ldots n_k$, there are natural numbers b and c such that $\beta(b, c, i) = n_i$ for all $i \leq k$.*

Proof. Let $m = max(k, n_0, \ldots, n_k)$ and $c = m!$. Let $u_i = 1 + (i + 1) \cdot c$. We claim that the u_i are pairwise coprime. Say $p|1+(i+1)c$ and $p|1+(j+1)c$ with $i < j \leq n$, then $p|(j-i)c$. Now if p were to divide c, then p would divide $(i+1)c$ and $1+(i+1)c$. Also, we have that $p \nmid j - i$ because $(j - i)|c = m!$. Therefore, u_i and u_j are coprime. Furthermore notice that $n_i < c < u_i$ for all i. Thus by the chinese remainder theorem there is $b < u_0 \cdot \ldots \cdot u_k$ such that $rm(u_i, b) = k_i$ for $i \leq k$ and $\beta(b, c, i) = rm(1 + (i + 1)c, b) = rm(u_i, b) = k_i$. □

Notice that the proof shows that there are computable functions which given $n_0, \ldots, n_k$ output b and c, respectively. Using the representability of the β function we can represent exponentiation as follows.

$$\begin{aligned} exp(x, y, z) \Leftrightarrow \exists u \exists v \, (&\varphi_\beta(u, v, 0, 1) \wedge \varphi_\beta(u, v, y, z) \wedge \\ &\forall w \, (w < y \rightarrow \exists v_1 \, (\exists v_2 \, (\varphi_\beta(u, v, w, v_1) \wedge \varphi_\beta(u, v, w+1, v_2) \wedge v_2 \equiv v_1 \cdot x)))) \,. \end{aligned}$$

Having seen that exponentiation is representable in the standard model we can without much more work represent Gödel numberings and their inverses. This allows us to represent Turing machines and thus all computable functions.

Lemma 5.3. *All computable functions are representable in arithmetic (in PA).*

We will not give a full proof of this theorem but instead just sketch the idea on how one could interpret Turing computations. Usually this theorem is proved by using partial recursive functions as the model of computation (see [Men09]) or register machines (see [EFT13]).

Proof idea. The idea of the proof is quite similar to the proof of the Entscheidungsproblem, Theorem 4.4. Recall that the computation of a Turing machine at a given time is specified by its Turing program, the contents of its tape, the state it is in and the position of its reading head.

All of these except the tape are finite objects so if we managed to code the tape in a finite object we could represent them as a sequence of natural numbers using a suitable Gödel numbering.

However, notice that we can in fact do that. Consider the program P_e running on input x at the start of its computation. It has $x + 1$ $1's$ on the tape, with the head on the first 1 and otherwise 0's. We can now represent the tape by a sequence of 3 binary strings, $l = 0$, $h = 1$, $r = \underbrace{1 \ldots 1}_{xtimes}0$. So we could specify this state by the tuple (q, l, h, r) where q is the Gödel number of the state q_0.

Now, given φ_e we can define a relation θ_e informally by saying

$$\theta_e(x,y) \Leftrightarrow \begin{array}{l}\text{there is a sequence of states which adheres to the Turing} \\ \text{program } P_e\text{, the first being a starting state for input } x, \\ \text{the last being a halting state and there are } y \text{ 1's in } l,h,r \\ \text{at this state}\end{array} \tag{20}$$

This definition needs to be carefully translated into the language of arithmetic. We will not do this here, hoping that the reader can see that this could be done with enough time and dedication. □

From our considerations for Lemma 5.3 we can conclude that given P_e we can obtain a formula θ_e such that $Th(\mathbb{N}) \vdash \theta_e$ iff $P_e(e)$ halts. Thus getting an m-reduction $K \leq_m \{i : Th(\mathbb{N}) \vdash \varphi_i\}$ where we take φ_i to be the i^{th} formula in a Gödel numbering of all S_{ar} formulas.

Corollary 5.4. *The standard model of arithmetic is undecidable. PA is undecidable.*

5.2. Gödel's first incompleteness theorem

Recall the set of symbols occuring in formulas in arithmetic:

$$0, 1, \cdot, +, \equiv, v_0, \ldots, (,), \neg, \vee, \wedge, \exists, \forall$$

We fix a Gödel numbering in similar fashion as the one for Turing machines:

1. $gn(0) = 2, gn(1) = 3, gn(+) = 5, gn(\cdot) = 7, \ldots, gn(\forall) = p_{11}$, and $gn(v_i) = p_{i+12}$,
2. for φ a formula with s_i being the i^{th} symbol of the formula, $gn(\varphi) = 2^{gn(s_0)} \cdot 3^{gn(s_1)} \ldots \cdot p_n^{gn(s_n)}$ where s_n is the last symbol of the formula,
3. for $\varphi_0\varphi_1 \ldots \varphi_n$ a sequent $gn(\varphi_0 \ldots \varphi_n) = 2^{gn(\varphi_0)} \ldots p_n^{gn(\varphi_n)}$,
4. and for a sequence of sequents $S_0 \ldots S_n$, $gn(S_0 \ldots S_n) = 2^{gn(S_0)} \cdot \ldots p_n^{gn(S_n)}$.

By unique prime factorization given a number n we can compute its prime factorization and thus obtain a computable inverse gn^{-1} for gn. Clearly, from the Church-Turing thesis we can computably tell whether n is the Gödel number of a formula. We can also computably check whether n is the Gödel number of a formula of the form of the axioms of PA and we can computably tell whether n is the Gödel number of a correct sequent. By Lemma 5.3 we get the following representations of the relations described in PA above:

1. $PA \vdash Form(\mathbf{m})$ iff $gn^{-1}(m)$ is an S^{ar} formula,
2. $PA \vdash Ax_{PA}(\mathbf{m})$ iff $gn^{-1}(m)$ is an axiom of PA,
3. $PA \vdash Seq_{PA}(\mathbf{m})$ iff $gn^{-1}(m)$ is a correct sequent of PA,

4. $PA \vdash Proof_{PA}(\mathbf{m}, \mathbf{n})$ iff $gn^{-1}(m)$ is a proof of $gn^{-1}(n)$.

Of course we also get the converse. Since the above formulas represent the correponding relations on natural numbers we get for example $PA \vdash \neg Proof_{PA}(\mathbf{m}, \mathbf{n})$ iff $gn^{-1}(m)$ is not a proof of $gn^{-1}(n)$.

Exercise 5.1. Show that $Proof_{PA}(m, n)$ is computable and thus representable in PA as claimed above.

Let T_{PA} be the set of Gödel numbers of all provable formulas in arithmetic, i.e.

$$T_{PA} = \{x : PA \vdash gn^{-1}(x)\}.$$

Theorem 5.5. *The set T_{PA} is c.e.*

Proof. Simply note that

$$x \in T_{PA} \text{ iff } \exists p Proof_{PA}(p, x).$$

□

Recall the following definitions:

Definition 5.2. A theory Φ in a symbol set S is *complete* if and only if $\Phi \vdash \varphi$ or $\Phi \vdash \neg\varphi$ for all sentences $\varphi \in L^S$. (cf. negation complete)

Definition 5.3. A theory Φ is *(computably) axiomatizable* if Ax_Φ is computable.

We can get relations Ax_Φ, Seq_Φ and $Proof_\Phi$ for any theory Φ. Clearly, if $\Phi \supseteq PA$ is axiomatizable, then Ax_Φ, $Form_\Phi$ and $Proof_\Phi$ are representable in Φ. To see this just notice that $\vdash$ is monotonic and that the relations are already representable in PA. We furthermore get the following fact.

Lemma 5.6. *If $\Phi \supseteq PA$ is complete and axiomatizable, then T_Φ is computable.*

Proof. If Φ is inconsistent then $T_\Phi = \mathbb{N}$, so it is trivially computable. If Φ is complete and consistent then for every formula φ either $gn(\varphi) \in T_\Phi$ or $gn(\neg\varphi) \in T_\Phi$ but not both. Therefore,

$$x \in \bar{T}_\Phi \text{ iff } gn(\neg gn^{-1}(x)) \in T_\Phi$$

and thus if Φ is axiomatizable it is c.e. As both T_Φ and $\bar{T}_\Phi$ are c.e., T_Φ is computable. □

Definition 5.4. Let Φ be a theory in the language of arithmetic. We say that Φ is ω-*consistent* if for each formula $\varphi(x)$ in S^{ar}, if $\Phi \vdash \exists x \neg\varphi(x)$, then there is $n \in \mathbb{N}$ such that $\Phi \nvdash \varphi(\mathbf{n})$.

Exercise 5.2. Show that ω-consistency implies consistency.

Proposition 5.7. *PA is ω-consistent.*

Proof. Note that $\mathbb{N} \models PA$. We prove the proposition by contraposition. Assume that $PA \vdash \varphi(\mathbf{m})$ for every $m \in \mathbb{N}$. Then $\mathbb{N} \models \neg\exists x\neg\varphi(x)$. As $\mathbb{N} \not\models \exists x\neg\varphi(x)$ we have that $PA \not\vdash \exists x\neg\varphi(x)$. □

Definition 5.5. A set S is *semi-representable* in a S^{ar} theory Φ if

$$x \in S \Leftrightarrow \Phi \vdash \varphi(\mathbf{x})$$

for φ a formula in arithmetic.

Theorem 5.8. *For each set S the following are equivalent.*

1. *S is c.e.*
2. *S is semi-representable in PA.*
3. *$S \leq_m T_{PA}$.*

Proof. (1) $\Rightarrow$ (2). Assume S is c.e., then it is Σ^0_1. Thus, let R be a computable relation such that

$$m \in S \Leftrightarrow \exists n R(m, n).$$

Assume R is represented in PA by ψ and let $\varphi(x) = \exists y\psi(x, y)$. We show that $m \in S$ iff $PA \vdash \varphi(\mathbf{m})$.

($\Rightarrow$). Say $m \in S$, and let n_0 such that $R(m, n_0)$. The following is a valid formula and thus provable in PA

$$\psi(\mathbf{m}, \mathbf{n}_0) \to \exists y\psi(\mathbf{m}, y)$$

and as R is representable we have that $PA \vdash \psi(\mathbf{m}, \mathbf{n}_0)$. Thus using the modus ponens rule we obtain $PA \vdash \exists y\psi(\mathbf{m}, y)$ and so $PA \vdash \varphi(\mathbf{m})$.

($\Leftarrow$). Say $PA \vdash \varphi(\mathbf{m})$. Hence $PA \vdash \exists y\neg\neg\psi(\mathbf{m}, y)$. By ω-consistency $PA \not\vdash \neg\psi(\mathbf{m}, \mathbf{n})$ for some n and therefore $R(m, n)$ must hold as ψ represents R. But then $\exists n R(m, n)$ and thus $m \in S$.

(2) $\Rightarrow$ (3). Assume S is semi-representable via φ. So $m \in S$ iff $PA \vdash \varphi(\mathbf{m})$. Then

$$m \in S \Leftrightarrow gn(\varphi(\mathbf{m})) \in T_{PA}.$$

Hence $S \leq_m T_{PA}$ as witnessed by the function $x \mapsto gn(\varphi(\mathbf{x}))$.

(3) $\Rightarrow$ (1). This is immediate from Theorem 5.5, Proposition 4.1 and Exercise 4.1.

□

Exercise 5.3. Show that if $\Phi \supseteq PA$ is axiomatizable and ω-consistent then for every set S, S is c.e. iff S is semi-representable in Φ iff $S \leq_m T_\Phi$.

Corollary 5.9. *S is representable in PA iff S is computable.*

We can now give a computability theoretic proof of Gödel's first incompleteness theorem.

Theorem 5.10 (Gödel's first incompleteness theorem)**.** *If Φ is an ω consistent axiomatizable extension of PA, then Φ is incomplete.*

Proof. Assume that Φ was complete, then T_Φ is computable by Lemma 5.6. But on the other hand $K \leq_m T_\Phi$ and thus K would be computable, a contradiction. $\square$

5.3. Gödel's second incompleteness theorem

Our proof deviated from the proof of Gödel in the last steps by using notions from computability theory which were not available when he initially proved his incompleteness theorems (in fact the computability theoretic notions were highly influenced by his proofs). To give a formal proof of his second incompleteness theorem we will go back to what he did and find a statement that is contradictory, i.e., a formula φ which says "I am not provable".

To obtain this we have to prove a theorem very similar to the recursion theorem in computability theory:

Remark. In what follows we will frequently use $\underset{\sim}{x}$ to denote the formal term for x, i.e., we will use $\underset{\sim}{x}$ to denote $\mathbf{x}$.

Lemma 5.11 (Fixed-point lemma)**.** *Suppose $\Phi \supseteq PA$. Then for every $\psi(x)$, there is a sentence φ such that*

$$\Phi \vdash \varphi \leftrightarrow \psi(\underset{\sim}{gn(\varphi)}).$$

Proof. Let $\psi \in L_1^{S_{ar}}$ and define f as

$$f(x) = \begin{cases} gn(\theta(\underset{\sim}{gn(\theta)})) & \text{if } gn(x) = \theta(y) \text{ is a formula} \\ 0 & \text{otherwise} \end{cases}$$

As f is computable it is representable in Φ, say by Γ_f. Fix $\theta \in L_1^{S_{ar}}$. Then

$$\Phi \vdash \forall y(\Gamma_f(\underset{\sim}{gn(\theta)}, y) \leftrightarrow y \equiv \underset{\sim}{gn(\theta(\underset{\sim}{gn(\theta)}))})$$

Let $\beta(z)$ be the formula defined as

$$\beta(z) = \forall y(\Gamma_f(z, y) \to \psi(y)).$$

Then $\Phi \vdash \beta(\underset{\sim}{gn(\theta)}) \leftrightarrow \forall y(y \equiv \underset{\sim}{gn(\theta(\underset{\sim}{gn(\theta)}))} \rightarrow \psi(y))$. Therefore

$$\Phi \vdash \beta(\underset{\sim}{gn(\theta)}) \leftrightarrow \psi(\underset{\sim}{gn(\theta(\underset{\sim}{gn(\theta)}))}))$$

Now, let $\varphi = \beta(\underset{\sim}{gn(\beta)})$, then

$$\Phi \vdash \beta(\underset{\sim}{gn(\beta)}) \leftrightarrow \psi(\underset{\sim}{gn(\beta(\underset{\sim}{gn(\beta)}))})$$

and hence $\Phi \vdash \varphi \leftrightarrow \psi(\underset{\sim}{gn(\varphi)}))$. $\square$

In Lemma 5.11 take $\psi = \forall y \neg Proof(y, x)$, then there is φ such that

$$\Phi \vdash \varphi \leftrightarrow \psi(\underset{\sim}{gn(\varphi)}). \tag{21}$$

Then φ intuitively expresses the property "I am not provable". And if $\Phi \vdash \varphi$, then Φ would prove that there is no proof of φ which is a contradiction if Φ was consistent. We have just shown the following.

Lemma 5.12. *If $\Phi \supseteq PA$ is consistent and axiomatizable then $\Phi \nvdash \varphi$.*

Using the ω-consistency of Φ we could get that Φ can not prove the negation of φ as well and thus obtain an alternative proof of the first incompleteness theorem.

Exercise 5.4. Prove Gödel's first incompleteness theorem using Lemma 5.11.

Furthermore notice that the consistency of a theory $\Phi \supseteq PA$ can be expressed by a Π^0_1 sentence,

$$Con_\Phi \Leftrightarrow \forall x \neg Proof_\Phi(x, gn(0 \equiv 1)).$$

Thus, Lemma 5.12 can be formalized as the sentence

$$Con_\Phi \rightarrow \forall x \neg Proof_\Phi(x, \underset{\sim}{gn(\varphi)}).$$

Using additional properties of $\vdash$ (see [Men09] for details) one can get that

$$\Phi \vdash Con_\Phi \rightarrow \forall x \neg Proof_\Phi(x, \underset{\sim}{gn(\varphi)}) \tag{22}$$

and using this we can prove the second incompleteness theorem.

Theorem 5.13 (Gödel's second incompleteness theorem)**.** *Let Φ be a consistent axiomatizable extension of PA. Then $\Phi \nvdash Con_\Phi$.*

Proof. Assume $\Phi \vdash Con_\Phi$, then by (22) $\Phi \vdash \forall x \neg Proof_\Phi(x, \underset{\sim}{gn(\varphi)})$. But notice that this formula is just $\psi(\underset{\sim}{gn(\varphi)})$ from (21) and thus $\Phi \vdash \varphi$, a contradiction. $\square$

5.4. Closing thoughts

In our proof of Theorem 5.10 we required ω-consistency. Rosser showed that it is possible to replace ω-consistency with consistency.

The incompleteness theorems are not unique to the language of arithmetic, they can be proven in any theory in which one can formalize a large enough part about arithmetic (in fact one only needs the semiring axioms, often also called Robinson arithmetic). Thus, in particular, they can be proved for ZFC which is considered to be the foundation of mathematics – the theory on which all of mathematics is developed. The second incompleteness theorem then shows that we can not prove the consistency of mathematics from within mathematics.

Combining the facts that if ZFC is consistent, then T_{ZFC} is Σ^0_1 complete (by Theorem 5.8; if ZFC is inconsistent then it is trivially computable) and the fact that the set arising from Hilbert's tenth problem H is Σ^0_1 complete as well one obtains the following fun fact.

If ZFC is consistent then there is a polynomial p for which we can not prove whether it has an integer root or not assuming that ZFC is consistent.

The idea behind this is as follows. Take a sentence φ such that by the first incompleteness theorem $ZFC \nvdash \varphi$ and $ZFC \nvdash \neg\varphi$. Then as both T_{ZFC} and H are Σ^0_1 complete, there is an m-reduction f from T_{ZFC} to H. So, look at $f(\varphi)$ and assume we could prove hat $f(\varphi) \in H$, then as f is computable we could also prove that $\varphi \in T_{ZFC}$, a contradiction. It also follows that we can prove that $f(\varphi)$ has an integer root iff ZFC is inconsistent.

Another interesting consequence concerns open problems in mathematics. For some famous open problems like for example the Riemann hypothesis we can show that if it is independent of ZFC, i.e. $ZFC \nvdash \varphi_R$ and $ZFC \nvdash \neg\varphi_R$ where φ_R is the formalization of the Riemann hypothesis, then the Riemann hypothesis is true.

Exercise 5.5. Show that if the Riemann hypothesis is neither provable nor refutable in ZFC, then it is true. *Hint: Look at the statement of the conjecture and see of which complexity it is. Assume that it was neither provable nor refutable and conclude that then there is an algorithmic solution to it.*

References

[Coo03] S. Barry Cooper. *Computability Theory.* CRC Press, 2003.

[EFT13] H.-D. Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical Logic.* Springer Science & Business Media, 2013.

[Men09] Elliott Mendelson. *Introduction to Mathematical Logic.* CRC press, 2009.

A. Partial recursive functions

We first define the primitive recursive functions.

Definition A.1 (Primitive recursive functions)**.**

1. The following functions, called initial functions are primitive recursive:
   a) The *zero* function $\mathbf{0} : n \mapsto 0$ for all $n \in \mathbf{N}$,
   b) the *successor function* $n' : n \mapsto n+1$ for all $n \in \mathbf{N}$,
   c) the *projection functions* $U_i^k : (n_0, \ldots, n_k) \mapsto n_i$ for all $i \leq k$, and $(n_0, \ldots, n_k) \in \mathbf{N}^k$.
2. If $g, h, h_0, \ldots, h_l$ are primitive recursive, then so is f defined by one of the following rules
   a) *Substitution*:
   $$f(\bar{m}) = g(h_0(\bar{m}), \ldots, h_l(\bar{m}))$$
   b) *Primitive Recursion*:
   $$f(\bar{m}, 0) = g(\bar{m}), \quad f(\bar{m}, n+1) = h(\bar{m}, n, f(\bar{m}, n)).$$

The primitive recursive functions are an interesting subclass of functions themselves, in particular they are all total. However not all total computable functions are primitive recursive. An example for the interested reader is the Ackermann function. Our main interest is in a larger class of functions, the partial recursive functions.

Definition A.2 (Partial recursive functions)**.** A function f is partial recursive if it is primitive recursive or defined using the μ or minimization operator from a partial recursive function, i.e.
$$f(\bar{x}) = \mu y[g(\bar{x}, y) = 0]$$
where g is partial recursive and
$$\mu m[g(\bar{x}, m) = 0 = m_0 \Leftrightarrow g(\bar{x}, m_0) = 0 \ \& \ \forall m < m_0[g(\bar{x}, m) \downarrow \neq 0].$$

Thesis 2 (Church's thesis)**.** A function is partial recursive iff it is intuitively computable.

While one could prove the two notions of computability to be equivalent we will just rely on Church's and Turing's thesis in this lecture. The interested reader can find a proof of the equivalence in TODO.

Example A.1. *Addition, $+$, is primitive recursive.*

Proof. Addition is easily defined by the following recursion scheme:
$$\begin{aligned} m + 0 &= m \\ m + (n+1) &= (m+n) + 1 = (m+n)' \end{aligned}$$

If we want to be completely formal we could write it as:

$$\begin{aligned} +(m,0) &= U_1^1(m) \\ +(m,n+1) = +(m,n)' &= U_3^3(m,n,f(m,n))'. \end{aligned}$$

□

Exercise A.1. Show that multiplication and exponentiation are primitive recursive.