Intermediate Logic

Lecture notes

Sandy Berkovski

Bilkent University
Fall 2010

Chapter 8

Computability

Historically, issues of computability derive from the problem of finding values of functions. A computing procedure serving such a purpose was labelled ‘algorithm’ (the word itself comes from the name of al-Khoresmi, a Baghdad mathematician of the ninth century). Instances of algorithms include such procedures as finding the greatest common divisor of two natural numbers, solving a system of two linear equations, and so forth. We can similarly talk of algorithms purporting to identify properties of the given relation. Thus we may look for an algorithm answering the question whether a certain natural number is prime, or whether certain two numbers are relatively prime.

8.1 The notion of an algorithm

Intuitively given instances of computing procedures indicate the following properties of the algorithm:

Unambiguous description. An algorithm must have such a description that would allow us to conduct it mechanically, without a need of attending to its mathematical (or logical, or other) content.

Determinism. Every step of the algorithm must be uniquely determined by its previous steps and the initial data.

Isolation. Computation must be conducted only by the algorithm’s instructions without any interference of external processes or computing devices.

With time the increase in the complexity of computing tasks lead to a situation where no concrete procedures were found. A hypothesis was raised that perhaps *no* algorithms exist for those tasks. The theoretical impossibility of finding such algorithms cannot be decided by any number of examples. It calls for a general solution. Therefore, a verdict on this problem cannot be reached unless we refine our intuitive notion of an algorithm and convert it into a mathematically precise concept.

8.2 Turing machines

There are several mathematically precise notions of algorithm currently available. All of them represent rather complex constructions. We shall now describe one such construction, called *Turing machine* (named after a British mathematician Alan Turing). First of all, we should note that, since algorithms have to be described unambiguously, the objects on which they operate—that is, the values of the functions, and the like—must also allow unambiguous descriptions. On the other hand, all unambiguously described mathematical objects must be represented as words of an alphabet. It is, then, natural to demand that algorithm operate with words of an alphabet. That is indeed how Turing machines work.

A Turing machine is a combination of several components. For the purposes of exposition we shall not try to achieve the highest possible rigour and abstractness. The four components are as follows:

1. The *tape* which is split into a finite number of squares. The tape has direction; accordingly, it contains the leftmost and rightmost squares. The tape may change in the process, so that new squares may be attached to it from the right. Each square has squares immediately to the right and to the left of it.

2. The scanning mechanism inspecting the content of the square called the *reading head.*

3. Two finite alphabets $A = \{\#, a_0, \dots, a_n\}$ and $Q = \{q_0, \dots, q_t\}$, where $n, t > 0$. The elements of A are the symbols of the machine, and the elements of Q are the *states* of the machine. In each moment of time the machine is present in some state, and each square contains a certain symbol. The symbol a_0 is 'blank', q_t is the final state, and q_1 is the initial state. The symbol # prevents the scanning device from exiting the tape. For convenience we shall let $\# = a_{-1}$.

4. The machine's programme is a finite set of words, conventionally called 'commands'. They have the form:
$$q_i a_j \rightarrowtail a_k C q_l,$$
where:
   (a) $q_i \neq q_n$ and $C \in \{L, R, H\}$;
   (b) Either $a_j = a_k = \#$, or else $a_j \neq \#$ and $a_k \neq \#$.

The above command would mean that every time the machine is in the state q_i and is scanning the square with the content a_j, its next step will be as follows:

1. In the scanned square the symbol a_j is deleted and replaced by a_k;
2. The reading head moves one square to the left if $C = L$, or to the right if $C = R$, or stays put if $C = H$;
3. The machine moves into the state q_l.

The correct computation must satisfy the following conditions:

1. Only the leftmost square contains #;
2. At the beginning and at the end the reading head scans the leftmost square;
3. At the beginning the machine is present in the initial state q_1, and at the end—in the final state q_0;
4. At the end all blank squares, if there are any, are located to the right of the non-blank squares.

We can now introduce a formal notion of correct computation. Let A^* be the set of all words of the alphabet A and let $A^+ \subseteq A^*$ be the set of all non-empty words.

Definition 8.1. A word of the form xq_iy is the *situation* of the Turing machine T, where $x \in A^*$, $y \in A^+$, $q_i \in Q$, and xy is representable as $\#z$, where z is #-free.

Intuitively the situation is a snapshot of the machine at any given moment: xy is the content of the tape, q_i is the state of the machine, whereas the first letter of y reflects the square currently scanned. The situation is called *initial* if it has the form $q_1\#t$. The situation is called *final* if it has the form $q_0ua_0^m$, where u is a_0-free. We can now refine the notion of machine command:

Definition 8.2. The situation s' is obtained from the situation s by applying the command K if one of the following conditions holds:

1. $K = q_ia_j \rightarrowtail a_kHa_l$, $s = vq_ia_jw$, $s' = vq_la_kw$;
2. $K = q_ia_j \rightarrowtail a_kLq_l$, $s = va_rq_ia_jw$, $s' = vq_la_ra_kw$;
3. $K = q_ia_j \rightarrowtail a_kLq_l$, $s = vq_ia_ja_rw$, $s' = va_kq_la_rw$;
4. $K = q_ia_j \rightarrowtail a_kLq_l$, $s = vq_ia_j$, $s' = va_kq_la_0$.

(The last clause evidently refers to the case when an extension of tape is required.)

The precise definition of correct Turing computation is as follows:

Definition 8.3. A sequence $\langle s_0, s_1, \dots, s_p\rangle$ is the *correct computation* by a Turing machine M if:

1. For every i, s_{i+1} is obtained from s_i by application of one of the commands of T;
2. s_0 is the initial state;
3. s_p is the final state.

If there is a correct computation of the machine M which starts with $q_1\#t_1a_0\cdots a_0t_n$ and finishes with the situation where $q_0\#ua_0^m$, then we say that M is processing the array $\langle t_1,\dots,t_n\rangle$ into the word u and we shall write $u = M[t_1,\dots,t_n]$. In this case we say that M is *applicable* to the array $[t_1,\dots,t_n]$. In general, M is applicable to the word $W \in A^*_{\text{in}}$ if $M[W]$ stops after a finite number of steps and there is $u \in A^*_{\text{out}}$ such that $u = M[W]$.

Let $A_{\text{in}}, A_{\text{out}} \subseteq A - \{a_{-1}, a_0\}$. We can now give a definition of the Turing-computable function:

Definition 8.4. A Turing machine M *computes* the function f of the arity n if f satisfies the following conditions:

1. The domain of f is contained in A^{*n}_{in}, while its range is in A^*_{out};
2. $\langle x_1,\dots,x_n\rangle \in A^{*n}_{\text{in}}$ just in case if $y = M[x_1,\dots,x_n]$, $y \in A^*_{\text{out}}$, then $y = f(x_1,\dots,x_n)$.

A function f is *Turing-computable* if there is a Turing machine computing f.

Analogously we may define the notion of a Turing-computable predicate:

Definition 8.5. A predicate $F(x_1,\dots,x_n)$ is *Turing-computable* if there is a Turing-computable function f such that:

$$f(x_1,\dots,x_n) = \begin{cases} 0 & \text{if } \langle x_1,\dots,x_n\rangle \in F \\ 1 & \text{otherwise.} \end{cases}$$

Let us now consider particular Turing machines.

Example 8.6. Let us build a machine which to each word x in the alphabet would attach the symbol a_1 from the right, so that $xa_1 = M[x]$:

$$q_1a_i \rightarrowtail a_iRq_1 \qquad q_1a_0 \rightarrowtail a_1Lq_2$$
$$q_2a_i \rightarrowtail a_iLq_2 \qquad q_2\# \rightarrowtail \#Hq_0,$$

where $i = -1, 1, \dots, n$. We shall check the correctness of our machine by letting $x = a_1a_2a_1$. We shall then obtain the following computation:

$$\begin{aligned}
&\#q_1a_2a_1a_2\\
&\#a_2q_1a_1a_2\\
&\#a_2a_1q_1a_2\\
&\#a_2a_1a_2q_1a_0\\
&\#a_2a_1q_2a_2a_1\\
&\#a_2q_2a_1a_2a_1\\
&\#q_2a_2a_1a_2a_1\\
&q_2\#a_2a_1a_2a_1\\
&q_0\#a_2a_1a_2a_1.
\end{aligned}$$

Example 8.7. Let us build a machine which doubles every word $x \in A^*_{\text{in}}$, or in other words, which computes the function $f(x) = xx$. If x is empty, then we may let $f(x) = x$.

$$\begin{array}{ll}
q_1\# \rightarrowtail \#Rq_1 & q_1a_i \rightarrowtail a_i'Rq_{1i}\\
q_{1i}a_j \rightarrowtail a_j'Rq_{1i} & q_{1i}a_i'' \rightarrowtail a_j''Rq_{1i}\\
q_{1i}a_0 \rightarrowtail a_i''Lq_2 & q_2a_i \rightarrowtail a_iLq_2\\
q_2a_i'' \rightarrowtail a_i''Lq_2 & q_2a_i' \rightarrowtail a_i'Rq_1\\
q_1a_i'' \rightarrowtail a_i''Rq_3 & q_3a_i'' \rightarrowtail a_i''Rq_3\\
q_3a_0 \rightarrowtail a_0Lq_4 & q_4a_i'' \rightarrowtail a_iLq_4\\
q_4a_i \rightarrowtail a_iLq_4 & q_4\# \rightarrowtail \#Hq_0\\
q_1a_0 \rightarrowtail a_0Lq_0, &
\end{array}$$

where $i, j = 1, \dots, n$. Here the principle is that every symbol of the input word is copied to the right. The copied symbol is marked to avoid its repeated copying and is 'remembered' with the state. Then the reading head moves to the right to reach the first empty square where the copy is being written. This copy is also marked (primed twice) to avoid repeated copying. Then the head moves to the left: the encounter with the primed symbol tells it to start the cycle all over again. If the cycle begins with the symbol which is marked twice, then the copying is finished. All that we have to do is to remove the marks. The final command purports to deal with an empty input word.

We should next consider the case where we need to build a machine computing the value of the functions defined on natural numbers and returning natural numbers as values. More precisely, we write ω to denote the set of natural numbers. Then we are interested in the functions $f\colon X \to \omega$, where $X \subseteq \omega^n$ for some n. Here we introduce a form of encoding natural numbers on the tape. We stipulate $A_{\text{in}} = \{|\}$ and represent the number m by a word consisting of m strokes. Such a word is sometimes called the *numeral* of the number m and is denoted by $\overline{m}$. It is then natural to define a function $\overline{f}\colon \{|\}^* \times \cdots \times \{|\}^* \to \{|\}$, such that:

$$f(x_1,\dots,x_n) = y \Longleftrightarrow \overline{f}(\overline{x_1},\dots,\overline{x_n}) = \overline{y}.$$

From the examples we have considered above it will then be clear how to build Turing machines $f(n) = n$, $f(n) = n+1$, and $f(n) = 2n$.

8.3 Recursive functions

The already mentioned function $f\colon X \to \omega$ $(X \subseteq \omega^n)$ will be called a *partial* function.

Definition 8.8. A partial function f^n is *computable* if there is an algorithm $\mathfrak{B}$ processing n-tuples $\alpha \in \omega^n$ and which does not process n-tuples $\alpha \notin X$ such that $\mathfrak{B}[\alpha] = f(\alpha)$, where $\alpha \in X$.

The following claim was put forward by Turing:

Proposition 8.9 (Turing's thesis). *Every computable partial function is Turing-computable.*

Proof. None. (Discussion...) □

We shall now attempt to further refine the notion of a computable function. We shall use a popular method of calculating the values of the function f of natural numbers (*i.e.* having the same form $f\colon X \to \omega$). The main idea here is in laying down the value $f(0)$ and fixing the *recursive* formula $f(n+1) = h(f(n))$ which allows to compute the value $f(n+1)$ if we already know the value $f(n)$. Such a process is generally called *recursion.* The issue of computing one function is then reduced to the issue of computing another.

Example 8.10. The function $f(n) = 2^n$ can be represented as:

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2^{n-1}\cdot 2 & \text{if } n > 0. \end{cases}$$

Then the function h will take the form $h(m) = m \cdot 2$.

An important fact to notice is that if a function is intuitively computable, then the function obtained from it by recursion will also be intuitively computable. In fact, we can generalise the notion of recursion a little further. The function $f(n+1)$ may depend not only on $f(n)$, but also on n. In this case we say that $f(n+1) = h(n, f(n))$. Thus, for $f(n) = n!$ (where $f(0) = 1$) we have $h(l,m) = m\cdot(l+1)$. So $(n+1)! = n!\cdot(n+1)$. Also, we may include a rule for computing n-ary functions. If we compute a function $f(x_1,\dots,x_k)$ and recursion is conducted on the variable x_k, then $f(x_1,\dots,x_{k-1},0) = g(x_1,\dots,x_{k-1})$, while $f(x_1,\dots,x_k+1) = h(x_1,\dots,x_{k-1},x_k,f(x_1,\dots,x_{k-1},x_k))$, where g^{k-1} and h^{k+1} are computable functions. (Intuitively, g expresses the initial condition, and h is the recursive step.)

Recursion is not unique in having a computability-preserving property. Another way is *superposition*, whereby our ability to compute $f(x)$ and $g(x)$ entails the ability to compute $g(f(x))$. Now, if we have a stock of simple functions which could be computed in a trivial way, then the functions obtained from them by recursion and superposition would also be computable. The *basic functions* include the following:

1. The zero function $Z(x) = 0$ for every x;
2. The successor function $S(x) = x+1$ for every x;
3. The projection function $I_i^n(x_1,\dots,x_n) = x_i$ for every $x_1,\dots,x_n$, where $1 \le i \le n$.

Definition 8.11. A function f is *primitive recursive* if either f is one of the basic functions, or else it is obtained from them by recursion and superposition.

Some instances of primitive recursive functions include:

1. $Z^n(x_1, \dots, x_n) = Z(I_i^n(x_1, \dots, x_n))$.
2. Constant functions of the form $k(x)$: $1(x) = S(Z(x))$, $2(x) = S(1(x))$, and so forth.
3. The function $f(x_1, x_2) = x_1 + x_2$ is obtained by recursion from $I_1^1(x_1)$ and $S(I_3^3(x_1, x_2, x_3)) = x_3 + 1$, so that we have:
$$\begin{cases} x_1 + 0 = I_1^1(x_1) \\ x_1 + (x_2 + 1) = (x_1 + x_2) + 1. \end{cases}$$
4. The function of limited subtraction:
$$x \dot{-} y = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{otherwise} \end{cases}$$
is obtained as follows:
$$\begin{cases} x \dot{-} 0 = x \\ x \dot{-} (y + 1) = (x \dot{-} y) \dot{-} 1. \end{cases}$$

Predicates are treated analogously. Let $F(x_1, \dots, x_n)$ be an n-ary predicate. We define the function χ_F^n as follows:
$$\chi_F(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } \langle x_1, \dots, x_n \rangle \in F \\ 0 & \text{if } \langle x_1, \dots, x_n \rangle \notin F. \end{cases}$$

The function χ_F is then called the *characteristic function* of the predicate F.

Definition 8.12. The predicate $F(x_1, \dots, x_n)$ defined on ω is *primitive recursive* if its characteristic function χ_F is primitive recursive.

[Men64, BJ89]

Chapter 9

Incompleteness results for arithmetic

9.1 Set-theoretic and semantic paradoxes

...

9.2 A Gödelian puzzle

We consider a machine M (perhaps a Turing machine) which prints out various expressions. Let the alphabet $A_{\text{out}} = \{\neg, P, N, (,)\}$. An expression $X \in A^*_{\text{out}}$ is *printable* if M prints it. The expression $X(X)$ is called *the norm* of the expression X. Clearly $X(X) \in A^*_{\text{out}}$. A *sentence* will be an expression of one of the four forms:

1. $P(X)$
2. $PN(X)$
3. $\neg P(X)$
4. $\neg PN(X)$.

We then let $V(P(X)) = \mathbf{1}$ iff X is printable, and $V(PN(X)) = \mathbf{1}$ iff the norm of X is printable. Similarly for the rest.

We now obtain a case of self-reference, since our machine M can print various sentences that say in effect what the machine can or cannot print. On the other hand, if M ever prints $P(X)$, then X is really printable. And if $PN(X)$ is printable, then $X(X)$ is also printable.

Suppose X is printable. Does it follow that $P(X)$ is printable? No. If X is printable, then $P(X)$ is true. But we have not shown that *all* true sentences must be printed by M. All we know is that M does not print any false sentences. Let us, therefore, ask whether M could possibly print all true sentences. The answer is again negative. There is at least one true sentence not printable by M. Consider the sentence $\neg PN(\neg PN)$. It is true just in case the norm of the expression $\neg PN$ is not printable. But the norm of $\neg PN$ is exactly $\neg PN(\neg PN)$. Hence $\neg PN(\neg PN)$ is true iff it is not printable. Since M prints no false sentences, the remaining option is that the sentence is true, but is not printable.

9.3 The abstract forms of Gödel's and Tarski's theorems

Let us consider a broad notion of mathematical theory to which Gödel's argument is applicable. To this end let us fix the components of the signature Σ of such a theory.

1. A countably infinite set $\mathscr{E}$ of expressions.
2. A set $\mathscr{S} \subseteq \mathscr{E}$ of sentences.
3. A set $\mathscr{P} \subseteq \mathscr{S}$ of provable sentences.
4. A set $\mathscr{R} \subseteq \mathscr{S}$ of refutable sentences.
5. A set $\mathscr{T} \subseteq \mathscr{S}$ of true sentences.

6. A set $\mathscr{H}$ of predicates. (Generally, predicates are expressions. Informally, they may be thought as sets of natural numbers.)

7. A function ϕ assigning to each expression E and every natural number n an expression $E(n)$ such that, for $H \in \mathscr{H}$, $H(n)$ is a sentence. (Informally, $H(n)$ says that the number n belongs to the set H.)

We say that a predicate H is true for $n \in \omega$ (or is *satisfied* by n) if $H(n) \in \mathscr{T}$. The set *expressed* by H is the set of all $n \in \omega$ satisfied by H.

Definition 9.1 (Expressibility)**.** Let X be a set of natural numbers. H *expresses* X iff for every number n, $H(n) \in \mathscr{T} \leftrightarrow n \in X$. The set X is called *expressible* in Σ if there is a predicate of Σ which expresses X.

Since the set $\mathscr{E}$ is countably infinite, there are at most denumerably many predicates of Σ. A question arises whether all sets of natural numbers would be expressible in Σ. That would be the case if there were denumerably many sets of natural numbers. That this is not so is a consequence of the following celebrated theorem. (Here the reader is encouraged to consult §2.2.2 for definitions of key notions.)

Proposition 9.2 (Cantor's theorem)**.** *The set $\wp(X)$ of all subsets of X has the cardinality strictly greater than the cardinality of X.*

Proof. First we show that $|X| \leq |\wp(X)|$. This is done by associating with each $x \in X$ a set $\{x\} \in \wp(X)$. This defines a one-to-one mapping of X into $\wp(X)$.

Suppose, for *reductio*, that $|X| = |\wp(X)|$ and that, therefore, there exists a one-to-one mapping f of X *onto* $\wp(X)$. Consider the set $M = \{x \in X \mid x \notin f(x)\}$. Clearly $M \subseteq X$. Then $M \in \wp(X)$. And then, by assumption, there must be $m \in X$ such that $f(m) = M$. Hence, on the one hand, if $m \in M$, then $m \notin f(m) = M$, and, on the other hand, if $m \notin M$, then $m \in f(m) = M$. We have obtained a contradiction. □

Now, let ω be the set of natural numbers. Then the set $\wp(\omega)$ of its subsets (*i.e.* the set of the sets of natural numbers) will have a cardinality greater than $|\omega|$. Therefore, there are more than denumerably many sets of natural numbers, and therefore, not every set of numbers will be expressible in Σ.

Consider now a theory T of the signature Σ.

Definition 9.3. A theory T is *correct* if its every sentence $A \in \Sigma$ provable in T is true, and every sentence $A \in \Sigma$ refutable in T is false.

In a correct theory, in other words, $\mathscr{P} \subseteq \mathscr{T}$, while $\mathscr{R} \cap \mathscr{T} = \varnothing$. We are now going to show that a correct theory T must contain a true sentence not provable in T.

9.3.1 The diagonal method

Let g be a one-to-one injection assigning to each expression E a number. That number $g(E)$ is called the *Gödel number* of E. We also assume that for every number there is an expression which has that number as its Gödel number. We further let E_n be an expression such that $g(E_n) = n$.

The *diagonalisation* of E_n is the expression $E_n(n)$. Given that E_n is a predicate, the sentence $E_n(n)$ is true iff the predicate E_n is satisfied by its own Gödel number n. Further, let $d(n)$ be the Gödel number of $E_n(n)$.

Definition 9.4 (Diagonal sets)**.** Let X be a set of numbers. The *diagonal set* X_d will be the set of all n such that $d(n) \in X$. In other words:

$$n \in X_d \leftrightarrow d(n) \in X.$$

9.3.2 Gödel's theorem: an abstract form

Definition 9.5. The set P is the set of Gödel numbers of all provable sentence:

$$x \in P \leftrightarrow x = g(A) \wedge A \in \mathscr{P}.$$

Definition 9.6. The *complement* of a set $X \subseteq \omega$ is the set $\widetilde{X}$ containing all natural numbers not in X:

$$\widetilde{X} = \{x \in \omega \mid x \notin X\}.$$

(Notice the deviation from our notation in Chapter 2.)

We are now ready to state Gödel's theorem in a fairly general form (*i.e.* without imposing specific restrictions on T).

Proposition 9.7 (After Gödel)**.** *If the set $\widetilde{P}_d$ is expressible in T and T is correct, then there is a sentence $A \in \Sigma$ such that $A \in \mathscr{T}$ and $A \notin \mathscr{P}$.*

Proof.

T is correct.	Ass.	(1)
$\widetilde{P}_d$ is expressible in T.	Ass.	(2)
$\forall n(H(n) \in \mathscr{T} \leftrightarrow n \in \widetilde{P}_d)$	Def. 9.1, (2)	(3)
$h = g(H)$	Ass.	(4)
$H(h) \in \mathscr{T} \leftrightarrow h \in \widetilde{P}_d$	UI, (3)	(5)
$h \in \widetilde{P}_d \leftrightarrow d(h) \in \widetilde{P} \leftrightarrow d(h) \notin P$	Def. 9.4, Def. 9.6	(6)
$d(h) = g(H(h))$	(4)	(7)
$d(h) \in P \leftrightarrow H(h) \in \mathscr{P}$	Def. 9.5, (7)	(8)
$d(h) \notin P \leftrightarrow H(h) \notin \mathscr{P}$	(8), PL	(9)
$H(h) \in \mathscr{T} \leftrightarrow H(h) \notin \mathscr{P}$	(5), (6), (9)	(10)
$(H(h) \in \mathscr{T} \wedge H(h) \notin \mathscr{P}) \vee (H(h) \notin \mathscr{T} \wedge H(h) \in \mathscr{P})$	(10), PL	(11)
$(H(h) \notin \mathscr{T} \wedge H(h) \in \mathscr{P}) \supset$ (T is not correct)	Def. 9.3	(12)
$\neg(H(h) \notin \mathscr{T} \wedge H(h) \in \mathscr{P})$	(12), (1), PL	(13)
$H(h) \in \mathscr{T} \wedge H(h) \notin \mathscr{P}$	(11), (13), PL	(14)

Therefore, $H(h)$ is the desired sentence. □

The following important notion is implicitly used in the above proof.

Definition 9.8 (Gödel sentences)**.** Let $X \subseteq \omega$. The *Gödel sentence* E_n for X is a sentence with the following property:

$$E_n \in \mathscr{T} \leftrightarrow n \in X.$$

(Informally, a Gödel sentence asserts that its own Gödel number lies in X.)

When considering a theory T (of the signature Σ) with specific properties it will be convenient to verify the claim that $\widetilde{P}_d$ is expressible in T by verifying three conditions:

G_1**:** For any set X expressible in T, the set X_d is expressible in T.

G_2**:** For any set X expressible in T, the set $\widetilde{X}$ is expressible in T.

G_3**:** The set P is expressible in T.

The three conditions jointly imply that $\widetilde{P}_d$ is expressible in T. It is in general the third condition which demands greater effort.

9.3.3 Tarski's theorem: an abstract form

A particularly simple route to Gödel's incompleteness theorems is offered by Tarski's theorem on the indefinability of truth. We shall now state it, again, without making specific assumptions about T. We first prove the following lemma.

Proposition 9.9 (Diagonal lemma)**.** *For any set X, if X_d is expressible in T, then there is a Gödel number for X.*

Proof.

H expresses X_d in T.	Ass.	(15)
$\forall n(H(n) \in \mathscr{T} \leftrightarrow n \in X_d)$	Def. 9.1, (15)	(16)
$h = g(H)$	Ass.	(17)
$d(h) = g(H(h))$	(17)	(18)
$H(h) \in \mathscr{T} \leftrightarrow h \in X_d$	UI, (16)	(19)
$h \in X_d \leftrightarrow d(h) \in X$	Def. 9.4, Def. 9.6	(20)
$H(h) \in \mathscr{T} \leftrightarrow d(h) \in X$	(19), (20)	(21)
$H(h)$ is a Gödel sentence for X.	(18), (21), Def. 9.8	□

Proposition 9.10. *If T satisfies the condition G_1, then for any X expressible in T, there is a Gödel number for X.*

Proof. If X is expressible in T, then, by G_1, X_d is expressible in T. Then, by the Diagonal lemma, there is a Gödel number for X. □

The abstract form of Gödel's theorem above is now allowed an even shorter proof.

Proof of Gödel's theorem based on the Diagonal lemma.

T is correct.	Ass.	(22)
$\widetilde{P}_d$ is expressible in T.	Ass.	(23)
There is a Gödel sentence G for $\widetilde{P}$.	(23), Diagonal lemma	(24)
$G \in \mathscr{T} \leftrightarrow G \notin \mathscr{P}$	(24), Def. 9.5	(25)
$G \in \mathscr{T} \wedge G \notin \mathscr{P}$	(22), (25), PL	□

The Diagonal lemma also yields Tarski's theorem.

Proposition 9.11 (After Tarski). *Let T be the set of Gödel numbers of the true sentences of T. Then the following claims hold:*

1. *The set $\widetilde{T}_d$ is not expressible in T.*
2. *If G_1 holds for T, then $\widetilde{T}$ is not expressible in T.*
3. *If G_1 and G_2 hold for T, then T is not expressible in T.*

Proof. We shall prove each of the claims in its own turn.

1. If $\widetilde{T}_d$ were expressible in T, then, by the Diagonal lemma, there must be a Gödel sentence for $\widetilde{T}$. But such a sentence would have been true iff its Gödel number was not the Gödel number of a true sentence. This is impossible; therefore, there is no Gödel sentence for $\widetilde{T}$. Hence, $\widetilde{T}_d$ is not expressible in T.
2. Suppose G_1 holds. Then, if $\widetilde{T}$ were expressible in T, $\widetilde{T}_d$ must be expressible in T. But this contradicts the just proven claim 1.
3. Suppose G_1 and G_2 hold. Then, if T were expressible in T, $\widetilde{T}$ must be expressible in T. But this contradicts the just proven claim 2. □

9.4 Tarski's theorem for arithmetic

There are several ways of proving incompleteness of Peano arithmetic—a standard axiomatisation of arithmetic. Closely following [Smu94], we are now interested in an especially simple proof utilising Tarski's theorem.

9.4.1 Syntax

We now move on to investigate a first-order arithmetical theory involving addition, multiplication, and exponentiation. The alphabet that we use includes the following thirteen symbols:

$$0 \quad ' \quad (\quad) \quad f \quad \prime \quad v \quad \neg \quad \supset \quad \forall \quad = \quad \leq \quad \sharp$$

The expressions 0, $0'$, $0''$, and so forth, are called *numerals* and will be used as names of the numbers 0, 1, 2, and so forth. It is then clear that $'$ serves as a name of the successor function. The symbols $f\prime$, $f\prime\prime$, and $f\prime\prime\prime$ are the names of the operations of addition, multiplication, and exponentiation. They are abbreviated as $+$, $\cdot$, and $\mathbf{E}$ respectively. The usual logical constants of the first-order calculus retain their meaning. But we also need a countably infinite list of variables $v_1, v_2, \ldots, v_n, \ldots$; these we put in our 13-symbol alphabet by abbreviating $v_1, v_2, v_3, \ldots$ as $v\prime, v\prime\prime, v\prime\prime\prime, \ldots$.

Terms are formed according to the following rules:

1. Every variable and numeral is a term.

2. If t_1 and t_2 are terms, then $(t_1 + t_2)$, $(t_1 \cdot t_2)$, and $(t_1\mathbf{E}t_2)$ are also terms.

A *constant* term contains no free variables.

An *atomic formula* is expression of the form $t_1 = t_2$ and $t_1 \leq t_2$. The set of formulae is formed according to the usual rules of first-order calculus.

The notions of *free* and *bound* variables and of *sentences* carry over from the first-order calculus. A special comment is needed for *substitution* of numerals for variables. For any number n, by $\underline{n}$ we mean the numeral designating n (this accords with our earlier notation for designating elements in a model). Thus, $\underline{5}$ abbreviates the expression $0'''''$. Further, deviating somewhat from our previous notation, we write $A(v_i)$ to indicate a formula having v_i as its only free variable. Similarly, for any numbers $k_1, \ldots, k_n$, we write $A(\underline{k_1}, \ldots, \underline{k_n})$ to mean a formula obtained by substituting the numerals $\underline{k_1}, \ldots, \underline{k_n}$ for the free occurrence of the variables $v_{i_1}, \ldots, v_{i_n}$. A formula is said to be *regular* if $i_1 = 1, \ldots, i_n = n$.

The *complexity* of formulae is interpreted analogously to the case of first-order calculus.

The following abbreviations will be used:

$$\begin{aligned}
(A \vee B) &\iff (\neg A \supset B)\\
(A \wedge B) &\iff \neg(A \supset \neg B)\\
A \leftrightarrow B &\iff ((A \supset B) \wedge (B \supset A))\\
\exists v_i A &\iff \neg\forall v_i \neg A\\
t_1 \neq t_2 &\iff \neg t_1 = t_2\\
t_1 < t_2 &\iff ((t_1 \leq t_2) \wedge (t_1 \neq t_2)\\
t_1^{t_2} &\iff t_1\mathbf{E}t_2\\
(\forall v_i \leq t)A &\iff \forall v_i(v_i \leq t \supset A)\\
(\exists v_i \leq t)A &\iff \neg(\forall v_i \leq t)\neg A.
\end{aligned}$$

We shall also omit outward parentheses, in accordance with our earlier usage. Another auxiliary notion is that of *designation*. It is defined according to the following rules:

1. A numeral $\underline{n}$ designates the number n.

2. If the constant terms c_1 and c_2 designate the numbers n_1 and n_2, then $(c_1 + c_2)$ designates the sum $n_1 \bar{+} n_2$, $(c_1 \cdot c_2)$ designates the product $n_1 \bar{\cdot} n_2$, $(c_1\mathbf{E}c_2)$ designates the number $n_1^{n_2}$, and $c_1{}'$ designates $n_1 + 1$.

9.4.2 The notion of truth

The definition of truth is carried out by induction on the complexity of formulae.

T_0**:** An atomic sentence $c_1 = c_2$ is true iff c_1 and c_2 designate the same natural numbers. An atomic sentence $c_1 \leq c_2$ is true iff c_1 designates the number less or equal than the number designated by c_2.

T_1**:** A sentence $\neg A$ is true iff A is not true.

T_2**:** A sentence $A \supset B$ is true iff either A is not true, or B is true.

T_3**:** A sentence $\forall v_i F$ is true iff for every $n \in \omega$, the sentence $F(\underline{n})$ is true.

An open formula $F(v_{i_1}, \ldots, v_{i_k})$ is *correct* if for all numbers $n_1, \ldots, n_k$, the sentence $F(\underline{n_1}, \ldots, \underline{n_k})$ is true.
Substitution and equivalence are defined in the usual way.

9.4.3 The notion of Arithmetic and arithmetic sets and relations

We can generalise our earlier notion of expressibility. We say that $F(v_1, \ldots, v_n)$ expresses the set of n-tuples $\langle k_1, \ldots, k_n \rangle$ (where $k_i \in \omega$). That is, $F(v_1, \ldots, v_n)$ *expresses* the relation $R(x_1, \ldots, x_n)$ just in case:

$$F(\underline{k_1}, \ldots, \underline{k_n}) \text{ is true } \leftrightarrow R(k_1, \ldots, k_n).$$

Example 9.12. The formula $\exists v_2(v_1 = 0'' \cdot v_2)$ expresses the set of even numbers.

A set or relation is called *Arithmetic* if it is expressed by a formula of $\mathscr{L}_E$. A set or relation is called *arithmetic* if it is expressed by a formula of $\mathscr{L}_E$ in which the exponential symbol does not occur. Analogously, a function $f(x_1, \ldots, x_n)$ is Arithmetic iff there is a formula $F(v_1, \ldots, v_n, v_{n+1})$ such that for all $x_1, \ldots, x_n, y \in \omega$, the sentence $F(\underline{x_1}, \ldots, \underline{x_n}, \underline{y})$ is true iff $f(x_1, \ldots, x_n) = y$.

9.4.4 Concatenation

We are looking to define the function $x *_b y$ for any $b > 1$.

Example 9.13. $42 *_{10} 8768 = 428768$.

We notice that $m *_{10} n = m \cdot 10^{\ell(n)} + n$. Hence, generally, $m *_b n = m \cdot b^{\ell(n)} + n$.

Proposition 9.14. *For every $b > 1$, the relation $x *_b y = z$ is Arithmetic.*

Proof. Omitted. □

Proposition 9.15. *For every $n, b > 1$, the relation $x_1 *_b \ldots *_b x_n = y$ is Arithmetic.*

Proof. By induction on n. □

9.4.5 Gödel numbering

We assign Gödel numbers to expressions in order to be able to talk about expressions 'indirectly' by talking about their Gödel numbers.

The idea of [Qui51] was to use the base 10 notation so that the expression $S_5S_3S_4$ were assigned the Gödel number 534. We use the base 13 notation and some modifications will be required. We use η, ϵ, and δ as digits for 10, 11, and 12 respectively. Thus we assign Gödel numbers to our thirteen symbols as follows:

0	$'$	$($	$)$	f	$\prime$	v	$\neg$	$\supset$	$\forall$	$=$	$\leq$	$\sharp$
1	0	2	3	4	5	6	7	8	9	η	ϵ	δ

Example 9.16. The Gödel number of the expression $0'' \leq 0'''$ is the number $100\epsilon1000_{13}$, that is, the number $0 + 0 \cdot 13 + 0 \cdot 13^2 + 1 \cdot 13^3 + 11 \cdot 13^4 + 0 \cdot 13^5 + 0 \cdot 13^6 + 1 \cdot 13^7$.

For any two expressions E_x and E_y the Gödel number of the expression E_xE_y is $x *_{13} y$.

Notice also that the numeral $\underline{n}$ consists of 0 followed by n primes, so that its Gödel number consists of 1 followed by n occurrences of n. Hence it is 13^n.

9.4.6 Tarski's theorem

The notions of Gödel sentence carries over from the earlier discussion.

Given any formula $F(v_1)$, the sentence $F(\underline{n})$ is equivalent to the sentence $\forall v_1(v_1 = \underline{n} \supset F(v_1))$. We shall designate $\forall v_1(v_1 = \underline{n} \supset F(v_1))$ as $F[\underline{n}]$. (The same abbreviation holds for any expression E which is not necessarily a formula.)

Let $e, n \in \omega$. Let the Gödel number of E be e. Then by $r(e, n)$ we mean the Gödel number of the expression $E[\underline{n}]$. Our goal is to show that $r(x, y)$ is Arithmetic. (The function $r(x, y)$ is also called the *representation function* of $\mathscr{L}_E$.)

Let $E_x[\underline{y}]$ be the expression $\forall v_1(v_1 = \underline{y} \supset E_x)$. Suppose the Gödel number of $\forall v_1(v_1 =$ is k. Then the Gödel number of $E_x[\underline{y}]$ is $k*13^y*8*x*3$. So we can see that $r(x, y)$ can be written as $\exists w(w = 13^y \wedge z = k*w*8*x*3$. Therefore:

Proposition 9.17. *The function $r(x, y)$ is Arithmetic.*

9.4.7 Diagonalisation and Tarski's theorem

Let the function $d(x) = r(x, x)$. The function d is called the *diagonal function.* Clearly it is Arithmetic, since r is Arithmetic. For any $n \in \omega$, $d(n)$ will be the Gödel number of $E_n[\underline{n}]$. We can now introduce the already familiar notion of the *diagonal set.* For any $X \subseteq \omega$, X_d is the set of n such that $d(n) \in X$.

Proposition 9.18. *If X is Arithmetic, then so is X_d.*

Proof. The set X_d consists of x such that $\exists y(d(x) = y \wedge y \in X)$. Since $d(x)$ is Arithmetic, there is $D(v_1, v_2)$ expressing the relation $d(x) = y$. But suppose $F(v_1)$ expresses X. Then X_d is expressed by $\exists v_2(D(v_1, v_2) \wedge F(v_2))$. □

We can now prove the following claim to be used (later ...) in constructing a true unprovable sentence:

Proposition 9.19. *If X is Arithmetic, then there is a Gödel sentence for X.*

Proof. Suppose X is Arithmetic. Then X_d is Arithmetic, too (by Proposition 9.18). Let $H(v_1) \in \mathscr{L}_E$ be a formula expressing X_d, and let h be its Gödel number. Then:

$$H(v_1) \text{ is true } \leftrightarrow h \in X_d \leftrightarrow d(h) \in X.$$

But $d(h)$ is the Gödel number of $H[\underline{h}]$. Therefore, $H[\underline{h}]$ is a Gödel sentence for X. □

We can now prove Tarski's theorem.

Proposition 9.20 (Tarski)**.** *The set T of Gödel numbers of the true Arithmetic sentences is not Arithmetic.*

Proof. We first show that $\widetilde{T}$ is not Arithmetic. Indeed, if $\widetilde{T}$ were Arithmetic, then, by Proposition 9.19, there would be a Gödel sentence for $\widetilde{T}$. But there is no Gödel sentence for $\widetilde{T}$, since such a sentence would be true if and only if it were not true. So $\widetilde{T}$ is not Arithmetic.

On the other hand, if $F(v_1)$ expresses X, then $\neg F(v_1)$ expresses $\widetilde{X}$. Hence, for every X, if X is Arithmetic, then $\widetilde{X}$ is Arithmetic, too. And since $\widetilde{T}$ is not Arithmetic, it now follows that T is not Arithmetic. □

Armed with Tarski's theorem, we will be able to argue that, since the set of Gödel numbers of all provable sentences is Arithmetic, truth and provability do not coincide.

9.5 Peano arithmetic

9.6 Gödel's theorems

9.7 Further notes on undecidability and incompleteness

[Smu94]