Gödel's Theorem

D. van Dalen

Logic Group
Preprint Series
No. 223
april 2003

ZENO Institute of Philosophy

ISBN 90-393-3418-8
ISSN 0929-0710

Introduction

The following pages make form a new chapter for the book *Logic and Structure*. This chapter deals with the incompleteness theorem, and contains enough basic material for the treatment of the required notions of computability, representability and the like.
This chapter will appear in the next edition of *Logic and Structure*. Comments are welcome.

Dirk van Dalen

7

Gödel's theorem

7.1 Primitive recursive functions

We will introduce a class of numerical functions which evidently are effectively computable. The procedure may seem rather ad hoc, but it gives us a surprisingly rich class of algorithms. We use the inductive method, that is, we fix a number of initial functions which are as effective as one can wish; after that we specify certain ways to manufacture new algorithms out of old ones.

The initial algorithms are extremely simple indeed: the successor function, the constant functions and the projection functions. It is obvious that composition (or substitution) of algorithms yields algorithms The use of recursion was as a device to obtain new functions already known to Dedekind'; that recursion produces algorithms from given algorithms is also easily seen. In logic the study of primitive recursive functions was initiated by Skolem, Herbrand, Gödel and others.

We will now proceed with a precise definition, which will be given in the form of an inductive definition. First we present a list of initial functions of an unmistakably algorithmic nature, and then we specify how to get new algorithms from old ones. All functions have their own arity, that is to say, they map $\mathbb{N}^k$ to $\mathbb{N}$ for a suitable k. We will in general not specify the arities of the functions involved, and assume that they are chosen correctly.
The so-called *initial functions* are

- the *constant functions* C_m^k with $C_m^k(n_0, \ldots, n_{k-1}) = m$,
- the *successor function* S with $S(n) = n + 1$,
- the *projection functions* P_i^k with $P_i^k(n_0, \ldots, n_{k-1}) = n_i$ $(i < k)$.

New algorithms are obtained from old ones by *substitution* or *composition* and *primitive recursion*,.

- A class $\mathcal{F}$ of functions is *closed under substitution* if $g, h_0, \ldots, h_{p-1} \in \mathcal{F} \Rightarrow$ $f \in \mathcal{F}$, where $f(\vec{n}) = g(h_0(\vec{n}), \ldots, h_{p-1}(\vec{n}))$

– $\mathcal{F}$ is *closed under primitive recursion* if $g, h \in \mathcal{F} \Rightarrow f \in \mathcal{F}$, where

$$\begin{cases} f(0, \vec{n}) = g(\vec{n}) \\ f(m+1, \vec{n}) = h(f(m, \vec{n}), \vec{n}, m). \end{cases}$$

Definition 7.1.1 *The class of primitive recursive functions is the smallest class of functions containing the constant functions, the successor function, and the projection functions, which is closed under substitution and primitive recursion.*

Remark. Substitution has been defined in a particular way: the functions that are substituted have all the same string of inputs. In order to make arbitrary substitutions one has to do a little bit of extra work. Consider for example the function $f(x,y)$ in which we want to substitute $g(z)$ for x and $f(z,x)$ for y: $f(g(z), f(z,x))$, This is accomplished as follows: put $h_0(x,z) = g(z) = g(P_1^2(x,z))$ and $h(x,z) = f(z,x) = f(P_1^2(x,z), P_0^2(x,z))$, Then the required $f(g(z), f(z,x))$ is obtained as $f(h_0(x,z), h_1(x,z))$. The reader is expected to handle cases of substitution that will come up.

Let us start by building up a stock of primitive recursive functions. The technique is not difficult at all, most readers will have used it at numerous occasions. The surprising fact is that so many functions can be obtained by these simple procedures. Here is a first example:
$x + y$, defined by

$$\begin{cases} x + 0 = x \\ x + (y+1) = (x+y) + 1 \end{cases}$$

We will reformulate this definition so that that the reader can see that it indeed fits the prescribed format:

$$\begin{cases} +(0,x) = P_0^1(x) \\ +(y+1,x) = S(P_0^3(+(y,x), P_0^2(x,y), P_1^2(x,y))). \end{cases}$$

As a rule we will we stick to traditional notations, so we will simply write $x + y$ for $+(y,x)$. We will also tacitly use the traditional abbreviations from mathematics, e.g. we will mostly drop the multiplication dot.

There are two convenient tricks to add or delete variables. The first one is the introduction of dummy variables.

Lemma 7.1.2 (dummy variables) *If f is primitive recursive, then so is g with $g(x_0, \ldots, x_{n-1}, z_0, \ldots, z_{m-1}) = f(x_0, \ldots, x_{n-1})$.*

Proof. Put $g(x_0, \ldots, x_{n-1}, z_0, \ldots, z_{m-1}) = f(P_0^{n+m}(\vec{x}, \vec{z}), \ldots, P_n^{n+m}(\vec{x}, \vec{z})$ □

Lemma 7.1.3 (identification of variables) *If f is primitive recursive, then so is $f(x_0, \ldots, x_{n-1})[x_i/x_j]$, where $i, j \leq n$*

Proof. We need only consider the case $i \neq j$. $f(x_0, \ldots, x_{n-1})[x_i/x_j] = f(P_0^n(x_0, \ldots, x_{n-1}), \ldots, P_i^n(x_0, \ldots, x_{n-1}), \ldots, P_i^n(x_0, \ldots, x_{n-1}), \ldots, P_{n-1}^n(x_0, \ldots, x_{n-1}))$, where the second P_i^n is at the jth entry. □

A more pedestrian notation is $f(x_0, \ldots, x_i, \ldots, x_i, \ldots x_{n-1})$.

Lemma 7.1.4 (permutation of variables) *If f is primitive recursive, then so is g with $g(x_0, \ldots, x_{n-1}) = f(x_0, \ldots, x_{n-1})[x_i, x_j/x_j, x_i]$, where $i, j \leq n$*

Proof. Use substitution and projection functions. □

From now on we will use the traditional informal notations, e.g. $g(x) = f(x, x, x)$, or $g(x, y) = f(y, x)$. For convenience we have used and will use, when no confusion can arise, the vector notation for strings of inputs.
The reader can easily verify that the following examples can be cast in the required format of the primitive recursive functions.

1. $x + y$
$$\begin{cases} x + 0 = x \\ x + (y + 1) = (x + y) + 1 \end{cases}$$
2. $x \cdot y$
$$\begin{cases} x \cdot 0 = 0 \\ x \cdot (y + 1) = x \cdot y + x \quad \text{(we use (1))} \end{cases}$$
3. x^y
$$\begin{cases} x^0 = 1 \\ x^{y+1} = x^y \cdot x \end{cases}$$
4. *predecessor function*
$$p(x) = \begin{cases} x - 1 \text{ if } x > 0 \\ 0 \quad \text{if } x = 0 \end{cases}$$
Apply recursion:
$$\begin{cases} p(0) = 0 \\ p(x + 1) = x \end{cases}$$
5. *cut-off subtraction (monus)*
$$x \dot{-} y = \begin{cases} x - y \text{ if } x \geq y \\ 0 \text{ else.} \end{cases}$$
Apply recursion:
$$\begin{cases} x \dot{-} 0 = x \\ x \dot{-} (y + 1) = p(x \dot{-} y) \end{cases}$$
6. *factorial function*
$$n! = 1 \cdot 2 \cdot 3 \cdots (n - 1) \cdot n.$$
7. *signum function*
$$\mathrm{sg}(x) = \begin{cases} 0 \text{ if } x = 0 \\ 1 \text{ otherwise} \end{cases}$$
Apply recursion:

$$\begin{cases} sg(0) = 0 \\ sg(x+1) = 1 \end{cases}$$

8. $\overline{\text{sg}}(x) = 1 \dot{-} \text{sg}(x)$.

$$\overline{\text{sg}}(x) = \begin{cases} 1 \text{ if } x = 0 \\ 0 \text{ otherwise} \end{cases}$$

9. $| x - y |$.
   Observe that $| x - y | = (x \dot{-} y) + (y \dot{-} x)$

10. $f(\vec{x}, y) = \sum_{i=0}^{y} g(\vec{x}, i)$, where g is primitive recursive.

$$\begin{cases} \sum_{i=0}^{0} g(\vec{x}, i) = g(\vec{x}, 0) \\ \sum_{i=0}^{y+1} g(\vec{x}, i) = \sum_{i=0}^{y} g(\vec{x}, i) + g(\vec{x}, y+1) \end{cases}$$

11. $\prod_{i=0}^{y} g(\vec{x}, i)$, idem.

12. If f is primitive recursive and π is a permutation of the set $\{0, \ldots, n-1\}$, then g with $g(\vec{x}) = f(x_{\pi 0}, \ldots, x_{\pi(n-1)})$ is also primitive recursive.
13. If $f(\vec{x}, y)$ is primitive recursive, so is $f(\vec{x}, k)$

The definition of primitive recursive functions by direct means is a worthwhile challenge, and the reader will find interesting cases among the exercises. For an efficient and quick access to a large stock of primitive recursive functions there are, however, techniques that cut a number of corners. We will present them here.

In the first place we can relate sets and functions by means of *characteristic functions*. In the setting of number theoretic functions, we define characteristic functions as follows: for $A \subseteq \mathbb{N}^k$ the characteristic function $K_A : \mathbb{N}^k \to \{0, 1\}$ of A is given by $\vec{n} \in A \Leftrightarrow K_A(\vec{n}) = 1$ (and hence $\vec{n} \notin A \Leftrightarrow K_A(\vec{n}) = 0$). Warning: in logic the characteristic function is sometimes defined with 0 and 1 interchanged. For the theory that does not make any difference. Note that a subset of $\mathbb{N}^k$ is also called a *k-ary relation*. When dealing with relations we tacitly assume that we have the correct number of arguments, e.g. when we write $A \cap B$ we suppose that A, B are subsets of the same $\mathbb{N}^k$.

Definition 7.1.5 *A relation R is primitive recursive if its characteristic function is so.*

Note that this corresponds to the idea of using K_R as a test for membership.

The following sets (relations) are primitive recursive:

1. $\emptyset : K_\emptyset(n) = 0$ for all n.
2. The set of even numbers, E:

$$\begin{cases} K_E(0) = 1 \\ K_E(x+1) = \overline{\text{sg}}(K_E(x)) \end{cases}$$

3. The equality relation: $K_=(x,y) = \overline{\text{sg}}(|\ x - y\ |)$
4. The order relation: $K_<(x,y) = \overline{\text{sg}}((x+1) \dot{-} y)$.

Lemma 7.1.6 *The primitive recursive relations are closed under* $\cup, \cap, ^c$ *and bounded quantification.*

Proof Let $C = A \cap B$, then $x \in C \Leftrightarrow x \in A \wedge x \in B$, so $K_C(x) = 1 \Leftrightarrow K_A(x) = 1 \wedge K_B(x) = 1$. Therefore we put $K_C(x) = K_A(x) \cdot K_B(x)$. Hence the intersection of primitive recursive sets is primitive recursive. For union take $K_{A\cup B}(x) = \text{sg}(K_A(x) + K_B(x))$, and for the complement $K_{A^c}(x) = \overline{\text{sg}}(K_A(x))$.

We say that R *is obtained by bounded quantification from* S if $R(\vec{n}, m) := Qx \leq mS(\vec{n}, x)$, where Q is one of the quantifiers $\forall, \exists$.

Consider the bounded existential quantification: $R(\vec{x}, n) := \exists y \leq nS(\vec{x}, y)$, then $K_R(\vec{x}, n) = \text{sg}(\sum_{y\leq n} K_S(\vec{x}, y))$, so if S is primitive recursive, then R is so.

The $\forall$ case is similar; it is left to the reader. □

Lemma 7.1.7 *The primitive recursive relations are closed under primitive recursive substitutions, i.e. if* $f_0, \ldots, f_{n-1}$ *and* R *are primitive recursive, then so is* $S(\vec{x}) := R(f_0(\vec{x}), \ldots, f_{n-1}(\vec{x}))$.

Proof $K_S(\vec{x}) = K_R(f_1(\vec{x}), \ldots, f_{n-1}(\vec{x}))$ □.

Lemma 7.1.8 (definition by cases) *Let* $R_1, \ldots, R_p$ *be mutually exclusive primitive recursive predicates, such that* $\forall \vec{x}(R_1(\vec{x}) \vee R_2(\vec{x}) \vee \cdots \vee R_p(\vec{x}))$ *and let* $g_1, \ldots, g_p$ *be primitive recursive functions, then* f *with*

$$f(\vec{x}) = \begin{cases} g_1(\vec{x}) \text{ if } R_1(\vec{x}) \\ g_2(\vec{x}) \text{ if } R_2(\vec{x}) \\ \quad\vdots \\ g_p(\vec{x}) \text{ if } R_p(\vec{x}) \end{cases}$$

is primitive recursive.

Proof If $K_{R_i}(\vec{x}) = 1$, then all the other characteristic functions yield 0, so we put $f(\vec{x}) = g_1(\vec{x}) \cdot K_{R_1}(\vec{x}) + \ldots + g_p(\vec{x}) \cdot K_{R_p}(\vec{x})$. □

The natural numbers are well-ordered, that is to say, each non-empty subset has a least element. If we can test the subset for membership, then we can always find this least element effectively. This is made precise for primitive recursive sets.

Some notation: $(\mu y)R(\vec{x}, y)$ stands for the least number y such that $R(\vec{x}, y)$ if there is one. $(\mu y < m)R(\vec{x}, y)$ stands for the least number $y < m$ such that $R(\vec{x}, y)$ if such a number exists; if not, we simply take it to be m.

Lemma 7.1.9 (bounded minimalization) *R is primitive recursive $\Rightarrow$ $(\mu y < m)R(\vec{x}, y)$ is primitive recursive.*

Proof Consider the following table:

R	$R(\vec{x},0)$	$R(\vec{x},1)$	, ...,	$R(\vec{x},i)$,	$R(\vec{x},i+1)$	, ...,	$R(\vec{x},m)$
K_R	0	0	...	1	0	...	1
g	0	0	...	1	1	...	1
h	1	1	...	0	0	...	0
f	1	2	...	i	i	...	i

In the first line we write the values of $K_R(\vec{x}, i)$ for $0 \leq i \leq m$, in the second line we make the sequence monotone, e.g. take $g(\vec{x}, i) = \operatorname{sg} \sum_{j=0}^{i} K_R(\vec{x}, j)$. Next we switch 0 and 1: $h(\vec{x}, i) = \overline{\operatorname{sg}} g(\vec{x}, i)$ and finally we sum the h : $f(\vec{x}, i) = \sum_{j=0}^{i} h(\vec{x}, j)$. If $R(\vec{x}, j)$ holds for the first time in i, then $f(\vec{x}, m-1) = i$, and if $R(\vec{x}, j)$ does not hold for any $j < m$, then $f(\vec{x}, m-1) = m$.
So $(\mu y < m)R(\vec{x}, y) = f(\vec{x}, m-1)$, and thus *bounded minimalization* yields a primitive recursive function. □

We put $(\mu y \leq m)R(\vec{x}, y) := (\mu y < m+1)R(\vec{x}, y)$.

Now it is time to apply our arsenal of techniques to obtain a large variety of primitive recursive relations and functions.

Theorem 7.1.10 *The following are primitive recursive:*

1. *The set of primes:*
   $Prime(x) \Leftrightarrow x$ *is a prime* $\Leftrightarrow x \neq 1 \wedge \forall yz \leq x(x = yz \rightarrow y = 1 \vee z = 1)$.
2. *The divisibility relation:*
   $x \mid y \Leftrightarrow \exists z \leq y(x \cdot z = y)$
3. *The exponent of the prime p in the factorisation of x:*
   $(\mu y \leq x)[p^y \mid x \wedge \neg\, p^{y+1} \mid x]$
4. *The 'nth prime' function:*
   $$\begin{cases} p(0) = 2 \\ p(n+1) = (\mu x \leq p(n)^n)(x \text{ is prime } \wedge x > p(n)). \end{cases}$$

Note that we start to count the prime numbers from zero, and we use the notation $p_n = p(n)$. So $p_0 = 2, p_1 = 3, p_2 = 5, \ldots$. The first prime is p_0, and the i*th* prime is p_{i-1}.

Proof. One easily checks that the defining predicates are primitive recursive by applying the above theorems □.

Coding of finite sequences

One of the interesting features of the natural number system is that it allows a fairly simple coding of pairs of numbers, triples, ..., and n-tuples in general. There are quite a number of these codings around, each having its own strong points. The two best known ones are those of Cantor and of Gödel. Cantor's coding is given in exercise ??, Gödel's coding will be used

here. It is based on the well-known fact that numbers have a unique (up to order) prime factorization.

The idea is to associate to a sequence $(n_0, \ldots, n_{k-1})$ the number $2^{n_0+1} \cdot 3^{n_1+1} \cdot \ldots \cdot p_i^{n_i+1} \cdot \ldots \cdot p_{k-1}^{n_{k-1}+1}$. The extra $+1$ in the exponents is to take into account that the coding has to show the zero's that occur in a sequence. From the prime factorization of a coded sequence we can effectively extract the original sequence. The way we have introduced these codes makes the coding unfortunately not a bijection, for example, 10 is not a coded sequence, whereas 6 is. This is not a terrible drawback; there are remedies, which we will not consider here.

Recall that, in the framework of set theory a sequence of length n is a mapping from $\{0, \ldots, n-1\}$ to $\mathbb{N}$, so we define the *empty sequence* as the unique sequence of length 0, i.e. the unique map from $\emptyset$ to $\mathbb{N}$, which is the empty function (i.e. set). We put the code of the empty sequence 1.

Definition 7.1.11 *1.* $Seq(n) := \forall p, q \leq n(Prime(p) \wedge Prime(q) \wedge q < p \wedge p \mid n \rightarrow q \mid n) \wedge n \neq 0$. **(sequence number)**
In words: n is a sequence number if it is a product of consecutive positive prime powers.

2. $lth(n) := (\mu x \leq n+1)[\neg\, p_x \mid n]$ **(length)**

3. $(n)_i = (\mu x < n)[p_i^x \mid n \wedge \neg\, p_{i+1}^x \mid n] \dot{-} 1$ **(decoding *or* projection)**
In words: the exponent of the *ith* prime in the factorisation of n, minus *1.* $(n)_i$ extracts the ith element of the sequence.

4. $n * m = n \cdot \prod_{i=0}^{\mathrm{lth}(m)-1} p_{\mathrm{lth}(n)+i}^{(m)_i+1}$. **(concatenation)**
In words: if m, n are codes of two sequences $\vec{m}, \vec{n}$, then the code of the concatenation of $\vec{m}$ and $\vec{n}$ is obtained by the product of n and the prime powers that one gets by 'moving up' all primes in the factorization of m by the length of n.

Remark: 1 is trivially a sequence number. The length function only yields the correct output for sequence numbers, e.g. $\mathrm{lth}(10) = 1$. Furthermore the length of 1 is indeed 0, and the length of a 1-tuple is 1.

Notation. We will use abbreviations for the iterated decoding functions: $(n)_{i,j} = ((n)_i)_j$, etc.
Sequence numbers are from now on written as $\langle n_0, \ldots, n_{k-1} \rangle$. So, for example, $\langle 5, 0 \rangle = 2^6 \cdot 3^1$. We write $\langle\rangle$ for the code of the empty sequence. The binary coding, $\langle x, y \rangle$, is usually called a *pairing function.*

So far we have used a straightforward form of recursion, each next output depends on the parameters and on the previous output. But already the *Fibonacci sequence* shows us that there are more forms of recursion that occur in practice:

$$\begin{cases} F(0) = 1 \\ F(1) = 1 \\ F(n+2) = F(n) + F(n+1) \end{cases}$$

The obvious generalization is a function, where each output depends on the parameters and all the preceding outputs. This is called *course of value recursion.*

Definition 7.1.12 *For a given function $f(y, \vec{x})$ its 'course of value' function $\bar{f}(y, \vec{x})$ is given by*

$$\begin{cases} \bar{f}(0, \vec{x}) = 1 \\ \bar{f}(y+1, \vec{x}) = \bar{f}(y \cdot \vec{x}) \cdot p_y^{f(y,\vec{x})+1}, \end{cases}$$

Example: if $f(0) = 1, f(1) = 0, f(2) = 7$, then $\bar{f}(0) = 1, \bar{f}(1) = 2^{1+1}, \bar{f}(2) = 2^{1+1} \cdot 3^1, \bar{f}(3) = 2^2 \cdot 3 \cdot 5^8 = \langle 1, 0, 7 \rangle$.

Lemma 7.1.13 *If f is primitive recursive, then so is $\bar{f}$.*

Proof. Obvious. □

Since $\bar{f}(n+1)$ 'codes' so to speak all information on f up to the nth value, we can use $\bar{f}$ to formulate course-of-value recursion.

Theorem 7.1.14 *If g is primitive recursive and $f(y, \vec{x}) = g(\bar{f}(y, \vec{x}), y, \vec{x})$, then f is primitive recursive.*

Proof We first define $\bar{f}$.

$$\begin{cases} \bar{f}(0, \vec{x}) = 1 \\ \bar{f}(y+1, \vec{x}) = \bar{f}(y, \vec{x}) * \langle g(\bar{f}(y, \vec{x}), y, \vec{x}) \rangle. \end{cases}$$

$\bar{f}$ is obviously primitive recursive. Since $f(y, \vec{x}) = (\bar{f}(y+1, \vec{x}))_y$ we see that f is primitive recursive. □

By now we have collected enough facts for future use about the primitive recursive functions. We might ask if there are more algorithms than just the primitive recursive functions. The answer turns out to be yes. Consider the following construction: each primitive recursive function f is determined by its definition, which consists of a string of functions $f_0, f_1, \ldots, f_{n-1} = f$ such that each function is either an initial function, or obtained from earlier ones by substitution or primitive recursion.

It is a matter of routine to code the whole definition into a natural number such that all information can be effectively extracted from the code (see [Hinman], p.34). The construction shows that we may define a function F such that $F(x, y) = f_x(y)$, where f_x is the primitive recursive function with code x. Now consider $D(x) = F(x, x) + 1$. Suppose that D is primitive recursive, so $D = f_n$ for a certain n, but then $D(n) = F(n, n) + 1 = f_n(n) + 1 \neq f_n(n)$. contradiction. It is clear, however, from the definition of D that it is effective, so we have indicated how to get an effective function which is not primitive recursive. The above result can also be given the following formulation: there

is no binary primitive recursive function $F(x,y)$ such that each unary primitive function is $F(n,y)$ for some n. In other words, the primitive functions cannot be primitive recursively enumerated.

The argument is in fact completely general; suppose we have a class of effective functions that can enumerate itself in the manner considered above, then we can always "diagonalize out of the class" by the D function. We call this '*diagonalization*'. The moral of this observation is that we have little hope of obtaining *all* effective functions in an effective way. The diagonalization technique goes back to Cantor, who introduced it to show that the reals are not denumerable. In general he used diagonalization to show that the cardinality of a set is less than the cardinality of its power set.

Exercises

1. If h_1 and h_2 are primitive recursive, then so are f and g, where
$$\begin{cases} f(0) = a_1 \\ g(0) = a_2 \\ f(x+1) = h_1(f(x), g(x), x) \\ g(x+1) = h_2(f(x), g(x), x) \end{cases}$$
2. Show that the Fibonacci series is primitive recursive, where
$$\begin{cases} f(0) = f(1) = 1 \\ f(x+2) = f(x) + f(x+1) \end{cases}$$
3. Let $[a]$ denote the integer part of the real number a (i.e. the greatest integer $\leq a$). Show that $[\frac{x}{y+1}]$, for natural numbers x and y, is primitive recursive.
4. Show that $max(x,y)$ and $min(x,y)$ are primitive recursive.
5. Show that the gcd (greatest commion divisor) and lcm (least common multiple) are primitive recursive.
6. Cantor's pairing function is given by $P(x,y) = \frac{1}{2}((x+y)^2 + 3x + 2y)$. Show that P is primitive recursive, and that P is a bijection of $\mathbb{N}^2$ onto $\mathbb{N}$ (Hint. Consider in the plane a walk along all lattice points as follows: $(0,0) \to (0,1) \to (1,0) \to (0,2) \to (1,1) \to (2,0) \to (0,3) \to (1,2) \to \ldots$). Define the 'inverses' L and R such that $P(L(z), R(z)) = z$ and show that they are primitive recursive.
7. Show $p_n \leq 2^{2^n}$. For more about bounds on p_n see [Smoryński 1980].

7.2 Partial Recursive Functions

Given the fact that the primitive recursive functions do not exhaust the numerical algorithms, we extend in a natural way the class of effective functions. As we have seen that an effective generation of all algorithms invariably brings us in conflict with diagonalization, we will widen our scope by allowing partial functions. In this way the conflicting situation $D(n) = D(n) + 1$ for a certain n only tells us that D is not defined for n.

In the present context functions have natural domains, i.e. sets of the form $\mathbb{N}^n (= \{(m_0, \ldots, m_{n-1}) \mid m_i \in \mathbb{N}\}$, so called Cartesian products), a partial function has a domain that is a subset of $\mathbb{N}^n$. If the domain is all of $\mathbb{N}^n$, then we call the function *total*.

Example: $f(x) = x^2$ is total, $g(x) = \mu y[y^2 = x]$ is partial and not total, ($g(x)$ is the square root of x if it exists).

The algorithms that we are going to introduce are called *partial recursive functions*; perhaps recursive, partial functions would have been a better name. However, the name has come to be generally accepted. The particular technique for defining partial recursive functions that we employ here goes back to Kleene. As before, we use an inductive definition; apart from clause R7 below, we could have used a formulation almost identical to that of the definition of the primitive recursive functions. Since we want a built-in universal function, that is a function that effectively enumerates the functions, we have to employ a more refined technique that allows explicit reference to the various algorithms. The trick is not esoteric at all, we simply give each algorithm a code number, called its *index*. We fix these indices in advance so that we can speak of the 'algorithm with index e yields output y on input $(x_0, \ldots, x_{n-1})$', symbolically represented as $\{e\}(x_0, \ldots, x_{n-1}) \simeq y$.

The heuristics of this 'index applied to input' is that an index is viewed as a description of an abstract machine that operates on inputs of a fixed arity. So $\{e\}(\mathbf{n}) \simeq m$ must be read as 'the machine with index e operates on $\mathbf{n}$ and yields output m'. It may very well be the case that the machine does not yield an output, in that case we say that $\{e\}(\mathbf{n})$ diverges. If there is an output, we say that $\{e\}(\mathbf{n})$ converges. That the abstract machine is an algorithm will appear from the specification in the definition below.

Note that we do not know in advance that the result is a function, i.e. that for each input there is at most one output. However plausible that is, it has to be shown. Kleene has introduced the symbol $\simeq$ for 'equality' in contexts where terms may be undefined. This happens to be useful in the study of algorithms that need not necessarily produce an output. The abstract machines above may, for example, get into a computation that runs on forever. For example, it might have an instruction of the form 'the output at n is the successor of the output at $n+1$'. It is easy to see that for no n an output can be obtained. In this context the use of the existence predicate would be useful, and $\simeq$ would be the $\equiv$ of the theory of partial objects (cf. Troelstra–van Dalen, 2.2). The convention ruling $\simeq$ is: if $t \simeq s$ then t and s are simultaneously defined and identical, or they are simultaneously undefined.

Definition 7.2.1 *The relation $\{e\}(\vec{x}) \simeq y$ is inductively defined by*

R1 $\{\langle 0, n, q\rangle\}(m_0, \ldots, m_{n-1}) \simeq q$
R2 $\{\langle 1, n, i\rangle\}(m_0, \ldots, m_{n-1}) \simeq m_i$, *for* $0 \leq i < n$
R3 $\{\langle 2, n, i\rangle\}(m_0, \ldots, m_{n-1}) \simeq m_i + 1$, *for* $0 \leq i < n$

R4 $\{\langle 3, n+4\rangle\}(p, q, r, s, m_0, \ldots, m_{n-1}) \simeq p$ *if* $r = s$
$\{\langle 3, n+4\rangle\}(p, q, r, s, m_0, \ldots, m_{n-1}) \simeq q$ *if* $r \neq s$
R5 $\{\langle 4, n, b, c_0, \ldots, c_{k-1}\rangle\}(m_0, \ldots, m_{n-1}) \simeq p$ *if there are* $q_0, \ldots, q_{k-1}$ *such that* $\{c_i\}(m_0, \ldots, m_{n-1}) \simeq q_i (0 \leq i < k)$ *and* $\{b\}(q_0, \ldots, q_{k-1}) \simeq p$
R6 $\{\langle 5, n+2\rangle\}(p, q, m_0, \ldots, m_{n-1}) \simeq S_n^1(p, q)$
R7 $\{\langle 6, n+1\rangle\}(b, m_0, \ldots, m_{n-1}) \simeq p$ *if* $\{b\}(m_0, \ldots, m_{n-1}) \simeq p$.

The function S_n^1 in R6 will be specified in the S_n^m theorem below.

Keeping the above reading of $\{e\}(\vec{x})$ in mind, we can paraphrase the schema's as follows:

R1 the machine with index $\langle 0, n, q\rangle$ yields for input $(m_0, \ldots, m_{n-1})$ output q (the *constant function*),
R2 the machine with index $\langle 1, n, i\rangle$ yields for input $\vec{m}$ output m_i (the *projection function* P_i^n),
R3 the machine with index $\langle 2, n, i\rangle$ yields for input $\vec{m}$ output $m_i + 1$ (the *successor function* on the ith argument),
R4 the machine with index $\langle 3, n+4\rangle$ tests the equality of the third and fourth argument of the input and yields the first argument in the case of equality, and the second argument otherwise (the *discriminator function*),
R5 the machine with index $\langle 4, n, b, c_0, \ldots, c_{k-1}\rangle$ first simulates the machines with index $c_0, \ldots, c_{k-1}$ with input $\vec{m}$, then uses the output sequence $(q_0, \ldots, q_{k-1})$ as input and simulates the machine with index b (*substitution*),
R7 the machine with index $\langle 6, n+1\rangle$ simulates for a given input $b, m_0, \ldots, m_{n-1}$, the machine with index b and input $m_0, \ldots, m_{n-1}$ (*reflection*).

Another way to view R7 is that it provides a *universal machine* for all machines with n-argument inputs, that is to say, it accepts as an input the indices of machines, and then simulates them. This is the kind of machine required for the diagonalization process. If one thinks of idealized abstract machines, then R7 is quite reasonable. One would expect that if indices can be 'deciphered', a universal machine can be constructed. This was indeed accomplished by Alan Turing, who constructed (abstractly) a so-called universal Turing machine.

The scrupulous might call R7 a case of cheating, since it does away with all the hard work one has to do in order to obtain a universal machine, for example in the case of Turing machines.

As $\{e\}(\vec{x}) \simeq y$ is inductively defined, everything we proved about inductively defined sets applies here. For example, if $\{e\}(\vec{x}) \simeq y$ is the case, then we know that there is a formation sequence (see page 9) for it. This sequence specifies how $\{e\}$ is built up from simpler partial recursive functions.
Note that we could also have viewed the above definition as an inductive definition of the set of indices (of partial recursive functions).

Lemma 7.2.2 *The relation* $\{e\}(\vec{x}) \simeq y$ *is functional.*

Proof. We have to show that $\{e\}$ behaves as a function, that is $\{e\}(\vec{x}) \simeq y$, $\{e\}(\vec{x}) \simeq z \Rightarrow y = z$. This is done by induction on the definition of $\{e\}$. We leave the proof to the reader.

The definition of $\{e\}(\mathbf{n}) \simeq m$ has a computational content, it tells us what to do. When presented with $\{e\}(\mathbf{n})$, we first look at e; if the first 'entry' of e if $0, 1 or 2$, then we compute the output via the corresponding initial function. If the first 'entry' is 3, then we determine the output 'by cases'. If the first entry is 4, we first do the subcomputations indicated by $\{c_i\}(\mathbf{m})$, then we use the outputs to carry out the subcomputation for $\{b\}(\mathbf{n})$. And so on.

If R7 is used in such a computation, we are no longer guaranteed that it will stop; indeed, we may run into a loop, as the following simple example shows.

From R7 it follows, as we will see below, that there is an index e such that $\{e\}(x) = \{x\}(x)$. To compute $\{e\}$ for the argument e we pass, according to R7, to the right-hand side, i.e. we must compute $\{e\}(e)$, since e was introduced by R7, we must repeat the transitions to the right hand side, etc. Evidently our procedure does not get us anywhere!

Conventions. The relation $\{e\}(\vec{x}) \simeq y$ defines a function on a domain, which is a subset of the 'natural domain', i.e. a set of the form $\mathbb{N}^n$. Such functions are called *partial recursive functions*; they are traditionally denoted by symbols from the Greek alphabet, φ, ψ, σ, etc. If such a function is total on its natural domain, it is called *recursive*, and denoted by a roman symbol, f, g, h, etc. The use of the equality symbol '=' is proper in the context of total functions. In practice we will however, when no confusion arises, often use it instead of '$\simeq$' . The reader should take care not to confuse formulas and partial recursive functions; it will always be clear from the context what a symbol stands for. Sets and relations will be denoted by roman capitals. When no confusion can arise, we will sometimes drop brackets, as in $\{e\}x$ for $\{e\}(x)$. Some authors use a 'bullet' notation for partial recursive functions: $e \bullet \vec{x}$. We will stick to 'Kleene brackets': $\{e\}(\vec{x})$.

The following terminology is traditionally used in recursion theory:

Definition 7.2.3 *1. If for a partial function φ $\exists y(\varphi(\vec{x}) = y)$, then we say that φ* converges *at $\vec{x}$, otherwise φ* diverges *at $\vec{x}$.*
2. If a partial function converges for all (proper) inputs, it is called total.
3. A total partial recursive function (sic!) will be called a recursive function.
4. A set (relation) is called recursive *if its characteristic function (which, by definition, is total) is recursive.*

Observe that it is an important feature of computations as defined in definition 7.2.1, that $\{e\}(\psi_0(\vec{n}), \psi_1(\vec{n}), \ldots, \psi_{k-1}(\vec{n}))$ diverges if one of its arguments $\psi_i(\mathbf{n})$ diverges. So, for example, the partial recursive function $\{e\}(x) - \{e\}(x)$ need not converge for all e and x, we first must know that $\{e\}(x)$ converges!

This feature is sometimes inconvenient and slightly paradoxical, e.g. in direct applications of the discriminator scheme $R4$, $\{\langle 3,4\rangle\}(\varphi(x),\psi(x),0,0)$ is undefined when the (seemingly irrelevant) function $\psi(x)$ is undefined.

With a bit of extra work, we can get an index for a partial recursive function that does *definition by cases* on partial recursive functions:

$$\{e\}(\vec{x}) = \begin{cases} \{e_1\}(\vec{x}) \text{ if } g_1(\vec{x}) = g_2(\vec{x}) \\ \{e_2\}(\vec{x}) \text{ if } g_1(\vec{x}) \neq g_2(\vec{x}) \end{cases}$$

for recursive g_1, g_2.

Define

$$\varphi(\vec{x}) = \begin{cases} e_1 \text{ if } g_1(\vec{x}) = g_2(\vec{x}) \\ e_2 \text{ if } g_1(\vec{x}) \neq g_2(\vec{x}) \end{cases}$$

by $\varphi(\vec{x}) = \{\langle 3,4\rangle\}(e_1, e_2, g_1(\vec{x}), g_2(\vec{x}))$. So $\{e\}(\vec{x}) = \{\psi(\vec{x})\}(\vec{x}) =$ [byR7] $\{\langle 6, n+1\rangle\}(\psi(\vec{x}), \vec{x})$. Now use R5 (substitution) to get the required index.

Since the primitive recursive functions form such a natural class of algorithms, it will be our first goal to show that they are included in the class of recursive functions.

The following important theorem has a neat machine motivation. Consider a machine with index e operating on two arguments x and y. Keeping x fixed, we have a machine operating on y. So we get a sequence of machines, one for each x. Does the index of each such machine depend in a decent way on x? The plausible answer seems 'yes'. The following theorem confirms this.

Theorem 7.2.4 (The S_n^m Theorem) *For every m, n with $0 < m < n$ there exists a primitive recursive function S_n^m such that*

$$\{S_n^m(e, x_0, \ldots, x_{m-1})\}(x_m, \ldots, x_{n-1}) = \{e\}(\vec{x}).$$

Proof. The first function, S_n^1, occurs in R6, we have postponed the precise definition, here it is:

$$S_n^1(e,y) = \langle 4, (e)_1 \dot{-} 1, e, \langle 0, (e)_1 \dot{-} 1, y\rangle, \langle 1, (e)_1 \dot{-} 1, 0\rangle, \ldots, \langle 1, (e)_1 \dot{-} 1, n \dot{-} 2\rangle\rangle.$$

Note that the arities are correct, $\{e\}$ has one argument more than the constant function and the projection functions involved.

Now $\{S_n^1(e,y)\}(\vec{x}) = z \Leftrightarrow$ there are $q_0 \cdots q_{n-1}$ such that

$$\begin{aligned} &\{\langle 0, (e)_1 \dot{-} 1, y\rangle\}(\vec{x}) = q_0 \\ &\{\langle 1, (e)_1 \dot{-} 1, 0\rangle\}(\vec{x}) = q_1 \\ &\quad \ldots\ldots \\ &\{\langle 1, (e)_1 \dot{-} 1, n \dot{-} 2\rangle\}(\vec{x}) = q_{n-1} \\ &\{e\}(q_0, \ldots, q_{n-1}) = z. \end{aligned}$$

By the clauses R1 and R2 we get $q_0 = y$ and $q_{i+1} = x_i$ $(0 \leq i \leq n-1)$, so $\{S_n^1(e,y)\}(\vec{x}) = \{e\}(y, \vec{x})$. Clearly, S_n^1 is primitive recursive.

The primitive recursive function S_n^m is obtained by applying S_n^1 m times. From our definition it follows that S_n^m is also recursive.

The S_n^m function allows us to consider some inputs as parameters, and the rest as proper inputs. This is a routine consideration in everyday mathematics: 'consider $f(x,y)$ as a function of y'. The logical notation for this specification of inputs makes use of the *lambda operator*. Say $t(x,y,z)$ is a term (in some language), then $\lambda x \cdot t(x,y,z)$ is for each choice of y,z the function $x \mapsto t(x,y,z)$. we say that y and z are parameters in this function. The evaluation of these lambda terms is simple: $\lambda x \cdot t(x,y,z)(n) = t(n,y,z)$. This topic belongs to the so-called *lambda-calculus*, for us the notation is just a convenient tool to express ourselves succinctly.

The S_n^m theorem expresses a *uniformity* property of the partial recursive functions. It is obvious indeed that, say for a partial recursive function $\varphi(x,y)$, each individual $\varphi(n,y)$ is partial recursive (substitute the constant n function for x), but this does not yet show that the index of $\lambda y \cdot \varphi(x,y)$ is in a systematic, uniform way computable from the index of φ and x. By the S_n^m theorem, we know that the index of $\{e\}(x,y,z)$, considered as a function of, say, z depends primitive recursively on x and y: $\{h(x,y)\}(z) = \{e\}(x,y,z)$.

We will see a number of applications of the S_n^m theorem.

Next we will prove a powerful theorem about partial recursive functions, that allows us to introduce partial recursive functions by inductive definitions, or by implicit definitions. Partial recursive functions can by this theorem be given as solutions of certain equations.

Example.

$$\varphi(n) = \begin{cases} 0 & \text{if } n \text{ is a prime, or 0, or 1} \\ \varphi(2n+1)+1 & \text{otherwise.} \end{cases}$$

Then $\varphi(0) = \varphi(1) = \varphi(2) = \varphi(3) = 0$, $\varphi(4) = \varphi(9) + 1 = \varphi(19) + 2 = 2$, $\varphi(5) = 0$, and , e.g. $\varphi(85) = 6$. Prima facie, we cannot say much about such a sequence. The following theorem of Kleene shows that we can always find a partial recursive solution to such an equation for φ.

Theorem 7.2.5 (The Recursion Theorem) *There exists a primitive recursive function* rc *such that for each* e *and* $\vec{x}$ $\{\mathrm{rc}(e)\}(\vec{x}) = \{e\}(\mathrm{rc}(e), \vec{x})$.

Let us note first that the theorem indeed gives the solution r of the following equation: $\{r\}(\vec{x}) = \{e\}(r, \vec{x})$. Indeed the solution depends primitive recursively on the given index e: $\{f(e)\}(x) = \{e\}(f(e), x)$. If we are not interested in the (primitive recursive) dependence of the index of the solution on the old index, we may even be content with the solution of $\{f\}(x) = \{e\}(f, x)$.

Proof. Let $\varphi(m,e,\vec{x}) = \{e\}(S_{n+2}^2(m,m,e), \vec{x})$ and let p be an index of φ. Put $\mathrm{rc}(e) = S_{n+2}^2(p,p,e)$, then

$$\begin{aligned} \{\mathrm{rc}(e)\}(\vec{x}) &= \{S_{n+2}^2(p,p,e)\}(\vec{x}) = \{p\}(p,e,\vec{x}) = \varphi(p,e,\vec{x}) \\ &= \{e\}(S_{n+2}^2(p,p,e), \vec{x}) = \{e\}(\mathrm{rc}(e), \vec{x}). \end{aligned}$$

□

As a special case we get the

Corollary 7.2.6 *For each e there exists an n such that $\{n\}(\vec{x}) = \{e\}(n, \vec{x})$.*

Corollary 7.2.7 *If $\{e\}$ is primitive recursive, then the solution of the equation $\{f(e)\}(\mathbf{x}) = \{e\}(f(e), \mathbf{x})$ given by the recursion theorem is also primitive recursive.*

Proof. Immediate from the explicit definition of the function rc. □

We will give a number of examples as soon as we have shown that we can obtain all primitive recursive functions. For then we have an ample stock of functions to experiment on. First we have to prove some more theorems.

The partial recursive functions are closed under a general form of minimalization, sometimes called *unbounded search*, which for a given recursive function $f(y, \vec{x})$ and arguments $\vec{x}$ runs through the values of y and looks for the first one that makes $f(y, \vec{x})$ equal to zero.

Theorem 7.2.8 *Let f be a recursive function, then $\varphi(\vec{x}) = \mu y[f(y, \vec{x}) = 0]$ is partial recursive.*

Proof. The idea is to compute consecutively $f(0, \mathbf{x}), f(1, \mathbf{x}), f(2, \mathbf{x}), \ldots$ until we find a value 0. This need not happen at all, but if it does, we will get to it. While we are computing these values, we keep count of the number of steps. This is taken care of by a recursive function. So we want a function ψ with index e, operating on y and $\mathbf{x}$, that does the job for us, i.e. a ψ that after computing a positive value for $f(y, \vec{x})$ moves on to the next input y and adds a 1 to the counter. Since we have hardly any arithmetical tools at the moment, the construction is rather roundabout and artificial.
In the table below we compute $f(y, \vec{x})$ step by step (the outputs are in the third row), and in the last row we compute $\psi(y, \vec{x})$ backwards , as it were.

y	0	1	2	3	...	$k-1$	k
$f(y, \vec{x})$	$f(0, \vec{x})$	$f(1, \vec{x})$	$f(2, \vec{x})$	$f(3, \vec{x})$	...	$f(k-1, \vec{x})$	$f(k, \vec{x})$
	2	7	6	12	...	3	0
$\psi(y, \vec{x})$	k	$k-1$	$k-2$	$k-3$	...	1	0

$\psi(0, \vec{x})$ is the required k. The instruction for ψ is simple:

$$\psi(y, \vec{x}) = \begin{cases} 0 \text{ if } f(y, \vec{x}) = 0 \\ \psi(y+1, \vec{x}) + 1 \text{ else} \end{cases}$$

In order to find an index for ψ, put $\psi(y, \vec{x}) = \{e\}(y, \vec{x})$ and look for a value for e. We introduce two auxiliary functions ψ_1 and ψ_2 with indices b and c such that $\psi_1(e, y, \vec{x}) = 0$ and $\psi_2(e, y, \vec{x}) = \psi(y+1, \vec{x}) + 1 = \{e\}(y+1, \vec{x}) + 1$. The index c follows easily by applying R_3, R_7 and the S_n^m-theorem. If $f(y, \vec{x}) = 0$ then we consider ψ_1, if not, ψ_2. Now we introduce, by clause R4, a new function χ_0 which computes an index:

$$\chi_0(e, y, \vec{x}) = \begin{cases} b \text{ if } f(y, \vec{x}) = 0 \\ c \text{ else} \end{cases}$$

and we put $\chi(e, y, \vec{x}) = \{\chi_0(e, y, \vec{x})\}(e, y, \vec{x})$. The recursion theorem provides us with an index e_0 such that $\chi(e_0, y, \vec{x}) = \{e_0\}(y, \vec{x})$.

We claim that $\{e_0\}(0, \vec{x})$ yields the desired value k, if it exists at all, i.e. e_0 is the index of the ψ we were looking for, and $\varphi(\vec{x}) = \{e\}(o, \vec{x})$.
If $f(y, \vec{x}) \neq 0$ then $\chi(e_0, y, \vec{x}) = \{c\}(e_0, y, \vec{x}) = \psi_2(e_0, y, \vec{x}) = \psi(y + 1, \vec{x}) + 1$, and if $f(y, \vec{x}) = 0$ then $\chi(e_0, y, \vec{x}) = \{b\}(e_0, y, \vec{x}) = 0$.

If, on the other hand, k is the first value y such that $f(y, \vec{x}) = 0$, then $\psi(0, \vec{x}) = \psi(1, \vec{x}) + 1 = \psi(2, \vec{x}) + 2 = \cdots = \psi(y_0, \vec{x}) + y_0 = k$. □

Note that the given function need not be recursive, and that the above argument also works for partial recursive f. We then have to reformulate $\mu y[f(x, \vec{y}) = 0]$ as the y such that $f(y, \vec{x}) = 0$ and for all $z < y$ $f(z, \vec{x})$ is defined and positive.

Lemma 7.2.9 *The predecessor is recursive.*

Proof. Define

$$x \dot{-} 1 = \begin{cases} 0 \text{ if } x = 0 \\ \mu y[y + 1 = x] \text{ else} \end{cases}$$

where $\mu y[y + 1 = x] = \mu y[f(y, x) = 0]$ with

$$f(y, x) = \begin{cases} 0 \text{ if } y + 1 = x \\ 1 \text{ else} \end{cases}$$ □

Theorem 7.2.10 *The recursive functions are closed under primitive recursion.*

Proof. Let g and h be recursive, and let f be given by

$$\begin{cases} f(0, \vec{x}) = g(\vec{x}) \\ f(y + 1, \vec{x}) = h(f(y, \vec{x}), \vec{x}, y). \end{cases}$$

We rewrite the schema as

$$f(y, \vec{x}) = \begin{cases} g(\vec{x}) \text{ if } y = 0 \\ h(f(y \dot{-} 1, \vec{x}), \vec{x}, y \dot{-} 1) \text{ otherwise.} \end{cases}$$

On the right hand side we have a definition by cases. So, it defines a partial recursive function with index, say, a of $y, \vec{x}$ and the index e of the function f we are looking for. This yields an equation $\{e\}(y, \vec{x}) = \{a\}(y, \vec{x}, e)$. By the recursion theorem the equation has a solution e_0. And an easy induction on y shows that $\{e_0\}$ is total, so f is a recursive function. □

We now get the obligatory

Corollary 7.2.11 *All primitive recursive functions are recursive.*

Now that we have recovered the primitive recursive functions, we can get lots of partial recursive functions.

Examples

1. define $\varphi(x) = \{e\}(x)+\{f\}(x)$, then by 7.2.11 and R5 φ is partial recursive and we would like to express the index of φ as a function of e and f. Consider $\psi(e,f,x) = \{e\}(x) + \{f\}(x)$. ψ is partial recursive, so it has an index n, i.e. $\{n\}(e,f,x) = \{e\}(x) + \{f\}(x)$. By the S_n^m theorem there is a primitive recursive function h such that $\{n\}(e,f,x) = \{h(n,e,f)\}(x)$. Therefore, $g(e,f) = h(n,e,f)$ is the required function.
2. There is a partial recursive function φ such that $\varphi(n) = (\varphi(n+1)+1)^2$: Consider $\{z\}(n) = \{e\}(z,n) = (\{z\}(n+1)+1)^2$. By the recursion theorem there is a solution rc(e) for z, hence φ exists. A simple argument shows that φ cannot be defined for any n, so the solution is the empty function (the machine that never gives an output).
3. *The Ackermann function*, see [Smorynski 1991], p.70 . Consider the following sequence of functions.

$$\begin{aligned}
&\varphi_0(m,n) = n+m\\
&\varphi_1(m,n) = n\cdot m\\
&\varphi_2(m,n) = n^m\\
&\quad\vdots\\
&\begin{cases}\varphi_{k+1}(0,n) = n\\ \varphi_{k+1}(m+1,n) = \varphi_k(\varphi_{k+1}(m,n),n) \quad (k\geq 2)\end{cases}
\end{aligned}$$

This sequence consists of faster and faster growing functions. We can lump all those functions together in one function

$$\varphi(k,m,n) = \varphi_k(m,n).$$

The above equations can be summarised

$$\begin{cases}\varphi(0,m,n) = n+m\\ \varphi(k+1,0,n) = \begin{cases}0 \text{ if } k=0\\ 1 \text{ if } k=1\\ n \text{ else}\end{cases}\\ \varphi(k+1,m+1,n) = \varphi(k,\varphi(k+1,m,n),n).\end{cases}$$

Note that the second equation has to distinguish cases according to the φ_{k+1} being the multiplication, exponentiation, or the general case $(k\geq 2)$. Using the fact that all primitive recursive functions are recursive, we rewrite the three cases into one equation of the form $\{e\}(k,m,n) = f(e,k,m,n)$ for a suitable recursive f (exercise 3). Hence, by the recursion theorem there exists a recursive function with index e that satisfies the equations above. Ackermann has shown that the function $\varphi(n,n,n)$ grows eventually faster than any primitive recursive function.
4. The recursion theorem can also be used for inductive definitions of sets or relations; this is seen by changing over to characteristic functions, e.g. suppose we want a relation $R(x,y)$ such that

$$R(x,y) \Leftrightarrow (x = 0 \wedge y \neq 0) \vee (x \neq 0 \wedge y \neq 0) \wedge R(x \dot{-} 1, y \dot{-} 1)).$$

Then we write

$$K_R(x,y) = \mathrm{sg}(\overline{\mathrm{sg}}(x) \cdot \mathrm{sg}(y) + \mathrm{sg}(x) \cdot \mathrm{sg}(y) \cdot K_R(x \dot{-} 1, y \dot{-} 1)),$$

so there is an e such that

$$K_R(x,y) = \{e\}(K_R(x \dot{-} 1, y \dot{-} 1), x, y).$$

Now suppose K_R has index z then we have

$$\{z\}(x,y) = \{e'\}(z,x,y).$$

The solution $\{n\}$ as provided by the recursion theorem is the required characteristic function. One immediately sees that R is the relation 'less than'. Therefore $\{n\}$ is total, and hence recursive; this shows that R is also recursive.Note that by the remark following the recursion theorem we even get the primitive recursiveness of R.

The following fundamental theorem is extremely useful for many applications. Its theoretical importance is that it shows that all partial recursive functions can be obtained from a primitive recursive relation by *one* minimalization.

So minimalization is the missing link between primitive recursive and (partial) recursive.

Theorem 7.2.12 (Normal Form Theorem). *There is a primitive recursive predicate T such that $\{e\}(\vec{x}) = ((\mu z)T(e, \langle \vec{x} \rangle, z))_1$.*

Proof. Our heuristics for partial recursive functions was based on the machine metaphor: think of an abstract machine with actions prescribed by the clauses R1 through R7. By retracing the index e of such a machine, we more or less give a computation procedure. It now is a clerical task to specify all the steps involved in such a 'computation'. Once we have accomplished this, we have made our notion of 'computation' precise, and from the form of the specification, we can immediately conclude that "c is the code of a computation" is indeed primitive recursive. We look for a predicate $T(e,u,z)$ that formalizes the heuristic statement 'z is a (coded) computation that is performed by a partial recursive function with index e on input u' (i.e. $\langle \vec{x} \rangle$). The 'computation' has been arranged in such a way that the first projection of z is its output.

The proof is a matter of clerical perseverance—not difficult, but not exciting either. For the reader it is better to work out a few cases by himself and to leave the rest, than to spell out the following details.

First two examples.

(1) The successor function applied to $(1,2,3)$:
$S^3_1(1,2,3) = 2 + 1 = 3$. Warning, here S^3_1 is used for the successor function

operating on the second item of the input string of length 3. The notation only used here.
The index is $e = \langle 2,3,1\rangle$, the input is $u = \langle 1,2,3\rangle$, and the step is the direct computation $z = \langle 3, \langle 1,2,3\rangle, \langle 2,3,1\rangle\rangle = \langle 3,u,e\rangle$

(2) The composition of projection and constant functions.
$P_2^3(C_0^2(7,0),5,1) = 1$.
By R5 the input of this function has to be a string of numbers, so we have to introduce a suitable input. The simplest solution is to use (7,0) as input and manufacture the remaining 5 and 1 out of them. So let us put $P_2^3(C_0^2(7,0),5,1) = P_2^3(C_0^2(7,0),C_5^2(7,0),C_1^2(7,0))$.
In order to keep the notation readable, we will use variables instead of the numerical inputs.
$\varphi(y_0,y_1) = P_2^3(C_0^2(y_0,y_1),C_5^2(y_0,y_1),C_1^2(y_0,y_1)) = P_2^3(C_0^2(y_0,y_1),x_1,x_2)$.
Let us first write down the data for the component functions:

	index	input	step
C_0^2	$\langle 0,2,0\rangle = e_0$	$\langle y_0,y_1\rangle = u$	$\langle 0,u,e_0\rangle = z_0$
$C_{x_1}^2$	$\langle 0,2,x_1\rangle = e_1$	$\langle y_0,y_1\rangle = u$	$\langle x_1,u,e_1\rangle = z_1$
$C_{x_2}^2$	$\langle 0,2,x_2\rangle = e_2$	$\langle y_0,y_1\rangle = u$	$\langle x_2,u,e_2\rangle = z_2$
P_2^3	$\langle 1,3,2\rangle = e_3$	$\langle 0,x_1,x_2\rangle = u'$	$\langle x_2,u',e_3\rangle = z_3$

Now for the composition:

	index	input	step
$f(y_0,y_1)$	$\langle 4,2,e_3,e_0,e_1,e_2\rangle = e$	$\langle y_0,y_1\rangle = u$	$\langle x_2,\langle y_0,y_1\rangle,e,z_3,\langle z_0,z_1,z_2\rangle\rangle = z$

As we see in this example, 'step' means the last step in the chain of steps that leads to the output. Now for an actual computation on numerical inputs, all one has to do is to replace y_0,y_1,x_1,x_2 by numbers and write out the data for $\varphi(y_0,y_1)$.

We have tried to arrange the proof in a readable manner by providing a running commentary.

The ingredients for, and conditions on, computations are displayed below. The index contains the information given in the clauses Ri. The computation codes the following items:

(1) the output
(2) the input
(3) the index
(4) subcomputations.

Note that z in the table below is the 'master number', i.e. we can read off the remaining data from z, e.g. $e = (z)_2$, $\mathrm{lth}(u) = (e)_1 = (z)_{2,1}$, and the output (if any) of the computation, $(z)_0$. In particular we can extract the 'master numbers' of the subcomputations. So, by decoding the code for a computation, we can effectively find the codes for the subcomputations, etc. This suggests a primitive recursive algorithm for the extraction of the total

'history' of a computation from its code. As a matter of fact, that is essentially the content of the normal form theorem.

	Index	*Input*	*Step*	*Conditions on Subcomputations*
	e	u	z	
R1	$\langle 0, n, q\rangle$	$\langle\vec{x}\rangle$	$\langle q, u, e\rangle$	
R2	$\langle 1, n, i\rangle$	$\langle\vec{x}\rangle$	$\langle x_i, u, e\rangle$	
R3	$\langle 2, n, i\rangle$	$\langle\vec{x}\rangle$	$\langle x_i + 1, u, e\rangle$	
R4	$\langle 3, n+4\rangle$	$\langle p, q, r, s, \vec{x}\rangle$	$\langle p, u, e\rangle$ if $r = s$ $\langle q, u, e\rangle$ if $r \neq s$	
R5	$\langle 4, n, b, c_0, \ldots, c_{k-1}\rangle$	$\langle\vec{x}\rangle$	$\langle (z')_0, u, e, z',$ $\langle z''_0, \ldots, z''_{k-1}\rangle\rangle$	$z', z''_0, \ldots, z''_{k-1}$ are computations with indices $b, c_0, \ldots, c_{k-1}$. z' has input $\langle (z''_0)_0, \ldots, (z''_{k-1})_0\rangle$.
R6	$\langle 5, n+2\rangle$	$\langle p, q, \vec{x}\rangle$	$\langle s, u, e\rangle$	(cf. 7.2.4)
R7	$\langle 6, n+1\rangle$	$\langle b, \vec{x}\rangle$	$\langle (z')_0, u, e, z'\rangle$	z' is a computation with input $\langle\vec{x}\rangle$ and index b.

We will now proceed in a (slightly) more formal manner, by defining a predicate $C(z)$ (for z is a computation), using the information of the preceding table. For convenience, we assume that in the clauses below, sequences u (in $Seq(u)$) have positive length.

$C(z)$ is defined by cases as follows:

$$
C(z) := \begin{cases}
\exists q, u, e < z[z = \langle q, u, e\rangle \wedge \mathrm{Seq}(u) \wedge e = \langle 0, \mathrm{lth}(u), q\rangle] & (1)\\
\text{or} & \\
\exists u, e, i < z[z = \langle (u)_i, u, e\rangle \wedge \mathrm{Seq}(u) \wedge e = \langle 1, \mathrm{lth}(u), i\rangle] & (2)\\
\text{or} & \\
\exists u, e, i < z[z = \langle (u)_i + 1, u, e\rangle \wedge \mathrm{Seq}(u) \wedge e = \langle 2, \mathrm{lth}(u), i\rangle] & (3)\\
\text{or} & \\
\exists u, e < z[\mathrm{Seq}(u) \wedge e = \langle 3, \mathrm{lth}(u)\rangle \wedge \mathrm{lth}(u) > 4 \wedge ([z = \langle (u)_0, u, e\rangle \wedge & \\
\quad \wedge (u)_2 = (u)_3] \vee [z = \langle (u)_1, u, e\rangle \wedge (u)_2 neq (u)_3])] & (4)\\
\text{or} & \\
\mathrm{Seq}(z) \wedge \mathrm{lth}(z) = 5 \wedge \mathrm{Seq}((z)_2) \wedge \mathrm{Seq}((z)_4) \wedge \mathrm{lth}((z)_2) = & \\
\quad = 3 + lth((z)_4) \wedge (z)_{2,0} = 4 \wedge C((z)_3) \wedge (z)_{3,0} = (z)_0 \wedge (z)_{3,1} = & \\
\quad = \langle (z)_{4,0,0}, \ldots, (z)_{4,\mathrm{lth}((z)_4),0}\rangle \wedge (z)_{3,2} = (z)_{2,2} \wedge & \\
\quad \wedge \bigwedge_{i=0}^{\mathrm{lth}((z)_4)-1} [C((z)_{4,i}) \wedge (z)_{4,i,2} = (z)_{0,2+i} \wedge (z)_{4,i,1} = (z)_1] & (5)\\
\text{or} & \\
\exists s, u, e < z[z = \langle s, u, e\rangle \wedge \mathrm{Seq}(u) \wedge e = \langle 5, \mathrm{lth}(u)\rangle \wedge & \\
\quad s = \langle 4, (u)_{0,1} \dot{-} 1, (u)_0, \langle 0, (u)_{0,1} \dot{-} 1, (u)_1\rangle, \langle 1, (u)_{0,1} \dot{-} 1, 0\rangle, \ldots & \\
\quad \ldots, \langle 1, (u)_{0,1} \dot{-} 1, (e)_1 \dot{-} 2\rangle\rangle], & (6)\\
\text{or} & \\
\exists u, e, w < z[\mathrm{Seq}(u) \wedge e = \langle 6, \mathrm{lth}(y)\rangle \wedge z = \langle (w)_0, u, e, w\rangle \wedge C(w) \wedge & \\
\quad \wedge (w)_2 = (u)_0 \wedge (w)_1 = \langle (u)_1, \ldots, (u)_{\mathrm{lth}(u)-1}\rangle] & (7)
\end{cases}
$$

We observe that the predicate C occurs at the right hand side only for smaller arguments, furthermore all operations involved in this definition of $C(z)$ are primitive recursive. We now apply the recursion theorem, as in the example on page 4, and conclude that $C(z)$ is primitive recursive.

Now we put $T(e, \mathbf{x}, z) := C(z) \wedge e = (z)_2 \wedge \langle \mathbf{x} \rangle = (z)_1$. So the predicate $T(e, \mathbf{x}, z)$ formalizes the statement 'z is the computation of the partial recursive function (machine) with index e operating on input $\langle \vec{x} \rangle$'. The output of the computation, if it exists, is $U(z) = (z)_0$; hence we have $\{e\}(\mathbf{x}) = (\mu z T(e, \mathbf{x}, z))_0$

For applications the precise structure of T is not important, it is good enough to know that it is primitive recursive. □

Exercises

1. Show that the empty function (that is the function that diverges for all inputs) is partial recursive. Indicate an index for the empty function.
2. Show that each partial recursive function has infinitely many indices.
3. Carry out the conversion of the three equations of the Ackermann function into one function, see p.227.

7.3 Recursively enumerable sets

If a set A has a recursive characteristic function, then this function acts as an effective test for membership. We can decide which elements are in A and which not. Decidable sets, convenient as they are, demand too much; it is usually not necessary to decide what is in a set, as long as we can generate it effectively. Equivalently, as we shall see, it is good enough to have an abstract machine that only accepts elements, and does not reject them. If you feed it an element, it may eventually show a green light of acceptance, but there is no red light for rejection.

Definition 7.3.1 *1. A set (relation) is (recursively)* decidable *if it is recursive.*

2. *A set is* recursively enumerable *(RE) if it is the domain of a partial recursive function.*
3. $W_e^k = \{\vec{x} \in \mathbb{N}^k | \exists y(\{e\}(\vec{x}) = y\}$, *i.e. the domain of the partial recursive function* $\{e\}$. *We call* e *the RE index of* W_e^k. *If no confusion arises we will delete the superscript.*

Notation: we write $\varphi(\vec{x}) \downarrow$ (resp. $\varphi(\vec{x}) \uparrow$) for $\varphi(\vec{x})$ converges (resp. $\varphi(\vec{x})$ diverges).

It is good heuristics to think of RE sets as being accepted by machines, e.g. if A_i is accepted by machine $M_i (i = 0, 1)$, then we make a new machine

that simulates M_0 and M_1 simultaneously, e.g. you feed M_0 and M_1 an input, and carry out the computation alternatingly – one step for M_0 and then one step for M_1, and so n is accepted by M if it is accepted by M_0 or M_1. Hence the union of two RE sets is also RE.

Example 7.3.2 *1.* $\mathbb{N}$ = *the domain of the constant 1 function.*
2. $\emptyset$ = *the domain of the empty function. This function is partial recursive, as we have already seen.*
3. Every recursive set is RE. Let A be recursive, put

$$\psi(\vec{x}) = \mu y[K_A(\vec{x}) = y \wedge y \neq 0]$$

Then $Dom(\psi) = A$.

The recursively enumerable sets derive their importance from the fact that they are effectively given, in the precise sense of the following theorem. Furthermore it is the case that the majority of important relations (sets) in logic are RE. For example the set of (codes of) provable sentences of arithmetic or predicate logic is RE. The RE sets represent the first step beyond the decidable sets, as we will show below.

Theorem 7.3.3 *The following statements are equivalent,* ($A \subseteq \mathbb{N}$)*:*

1. $A =$ $Dom(\varphi)$ *for some partial recursive* φ,
2. $A =$ $Ran(\varphi)$ *for some partial recursive* φ,
3. $A = \{x | \exists y R(x, y)\}$ *for some recursive* R.

Proof. (1) $\Rightarrow$ (2). Define $\psi(x) = x \cdot \mathrm{sg}(\varphi(x)+1)$. If $x \in \mathrm{Dom}(\varphi)$, then $\psi(x) = x$, so $x \in \mathrm{Ran}(\psi)$, and if $x \in \mathrm{Ran}(\psi)$, then $\varphi(x)\downarrow$, so $x \in \mathrm{Dom}(\varphi)$.

(2) $\Rightarrow$ (3) Let $A = \mathrm{Ran}(\varphi)$, with $\{g\} = \varphi$, then

$$x \in A \Leftrightarrow \exists w[T(g, (w)_0, (w)_1) \wedge x = (w)_{1,0}].$$

The relation in the scope of the quantifier is recursive.
Note that w 'simulates' a pair: first co-ordinate—input, second co-ordinate—computation, all in the sense of the normal form theorem.

(3) $\Rightarrow$ (1) Define $\varphi(x) = \mu y R(x, y)$. φ is partial recursive and $Dom(\varphi) = A$.
Observe that (1) $\Rightarrow$ (3) also holds for $A \subseteq \mathbb{N}^k$. □

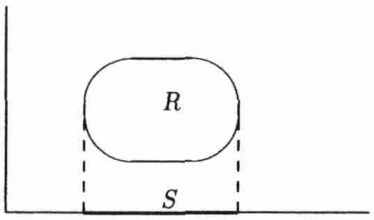
Since we have defined recursive sets by means of characteristic functions, and since we have established closure under primitive recursion, we can copy all the closure properties of primitive recursive sets (and relations) for the recursive sets (and relations).

Next we list a number of closure properties of RE-sets. We will write sets and relations also as predicates, when that turns out to be convenient.

Theorem 7.3.4 *1. If A and B are RE, then so are $A \cup B$ and $A \cap B$*
2. If $R(x, \vec{y})$ is RE, then so is $\exists x R(x, \vec{y})$

3. *If* $R(x,\vec{y})$ *is RE and* φ *partial recursive, then* $R(\varphi(\vec{y},\vec{z}),\vec{y})$ *is RE*
4. *If* $R(x,\vec{y})$ *is RE, then so are* $\forall x < zR(x,\vec{y})$ *and* $\exists x < zR(x,\vec{y})$.

Proof. (1) There are recursive R and S such that
$A\vec{y} \Leftrightarrow \exists x R(x,\vec{y})$,
$B\vec{y} \Leftrightarrow \exists x S(x,\vec{y})$.
Then
$$A\vec{y} \wedge B\vec{y} \Leftrightarrow \exists x_1 x_2 (R(x_1,\vec{y}) \wedge S(x_2,\vec{y})) \Leftrightarrow \exists z (R((z)_0,\vec{y}) \wedge S((z)_1,\vec{y})).$$
The relation in the scope of the quantifier is recursive, so $A \cap B$ is RE. A similar argument establishes the recursive enumerability of $A \cup B$. The trick of replacing x_1 and x_2 by $(z)_0$ and $(z)_1$ and $\exists x_1 x_2$ by $\exists z$ is called *contraction of quantifiers.*
(2) Let $R(x,\vec{y}) \Leftrightarrow \exists z S(z,x,\vec{y})$ for a recursive S, then $\exists x R(x,\vec{y}) \Leftrightarrow$ $\exists x \exists z S(z,x,\vec{y}) \Leftrightarrow \exists u S((u)_0,(u)_1,\vec{y})$. So the *projection* $\exists x R(x,\vec{y})$ of R is RE. Geometrically speaking, $\exists x R(x,\vec{y})$ is indeed a projection. Consider the two-dimensional case,



The vertical projection S of R is given by $Sx \Leftrightarrow \exists y R(x,y)$.
(3) Let R be the domain of a partial recursive ψ, then $R(\varphi(\vec{y},\vec{z}),\vec{y})$ is the domain of $\psi(\varphi(\vec{y},\vec{z}),\vec{y})$.
(4) Left to the reader. □

Theorem 7.3.5 *The graph of a partial function is RE iff the function is partial recursive.*

Proof. $G = \{(\vec{x},y) | y = \{e\}(\vec{x})\}$ is the graph of $\{e\}$. Now $(\vec{x},y) \in G \Leftrightarrow$ $\exists z (T(e,\langle\vec{x}\rangle,z) \wedge y = (z)_0)$, so G is RE.
Conversely, if G is RE, then $G(\vec{x},y) \Leftrightarrow \exists z R(\vec{x},y,z)$ for some recursive R. Hence $\varphi(\vec{x}) = (\mu w R(\vec{x},(w)_0,(w)_1))_0$, so φ is partial recursive. □

We can also characterize recursive sets in terms of RE-sets. Suppose both A and its complement A^c are RE, then (heuristically) we have two machines enumerating A and A^c. Now the test for membership of A is simple: turn both machines on and wait for n to turn up as output of the first or second machine. This must necessarily occur in finitely many steps since $n \in A$ or $n \in A^c$ (principle of the excluded third!). Hence, we have an effective test. We formalize the above:

Theorem 7.3.6 *A is recursive* $\Leftrightarrow$ *A and* A^c *are RE.*

Proof. $\Rightarrow$ is trivial: $A(\vec{x}) \Leftrightarrow \exists y A(\vec{x})$, where y is a dummy variable. Similarly for A^c.
$\Leftarrow$ Let $A(\vec{x})) \Leftrightarrow \exists y R(\vec{x}, y), \neg A(\vec{x}) \Leftrightarrow \exists z S(v, z)$. Since $\forall \vec{x}(A(\vec{x}) \vee \neg A(\vec{x}))$, we have $\forall \vec{x} \exists y(R(\vec{x}, y) \vee S(\vec{x}, y))$, so $f(\vec{x}) = \mu y[R(\vec{x}, y) \vee S(\vec{x}, y)]$ is recursive and if we plug the y that we found in $R(\vec{x}, y)$, then we know that if $R(\vec{x}, f(\vec{x}))$ is true, the $\vec{x}$ belongs to A. So $A(\vec{x}) \Leftrightarrow R(\vec{x}, f(\vec{x}))$, i.e. A is recursive. □

For partial recursive functions we have a strong form of definition by cases:

Theorem 7.3.7 *Let $\psi_1, \ldots, \psi_k$ be partial recursive, $R_0, \ldots, R_{k-1}$ mutually disjoint RE-relations, then the following function is partial recursive:*

$$\varphi(\vec{x}) = \begin{cases} \psi_0(\vec{x}) & \text{if } R_0(\vec{x}) \\ \psi_1(\vec{x}) & \text{if } R_1(\vec{x}) \\ \quad \vdots & \\ \psi_{k-1}(\vec{x}) & \text{if } R_{k-1}(\vec{x}) \\ \quad \uparrow & \text{else} \end{cases}$$

Proof. We consider the graph of the function φ.

$$G(\vec{x}, y) \Leftrightarrow (R_0(\vec{x}) \wedge y = \psi_1(\vec{x})) \vee \cdots \vee (R_{k-1}(\vec{x}) \wedge y = \psi_{k-1}(\vec{x})).$$

By the properties of RE-sets, $G(\vec{x}, y)$ is RE and, hence, $\varphi(\vec{x})$ is partial recursive. (Note that the last case in the definition of φ is just a bit of decoration).□

Now we can show the existence of undecidable RE sets.

Examples

(1) **(The Halting Problem (Turing))**
Consider $K = \{x | \exists z T(x, x, z)\}$. K is the projection of a recursive relation, so it is RE. Suppose that K^c is also RE, then $x \in K^c \Leftrightarrow \exists z T(e, x, z)$ for some index e. Now $e \in K \Leftrightarrow \exists z T(e, e, z) \Leftrightarrow e \in K^c$. Contradiction. Hence K is not recursive by theorem 7.3.6. This tells us that there are recursively enumerable sets which are not recursive. In other words, the fact that one can effectively enumerate a set, does not guarantee that it is decidable.

The decision problem for K is called the *halting problem*, because it can be paraphrased as 'decide if the machine with index x performs a computation that halts after a finite number of steps when presented with x as input. Note that it is *ipso facto* undecidable if 'the machine with index x eventually halts on input y'.
It is a characteristic feature of decision problems in recursion theory, that they concern tests for inputs out of some domain. It does not make sense to ask for a decision procedure for, say, the Riemann hypothesis, since there trivially is a recursive function f that tests the problem in the sense that $f(0) = 0$ if the Riemann hypothesis holds and $f(0) = 1$ if the Riemann hypothesis is

false. Namely, consider the functions f_0 and f_1, which are the constant 0 and 1 functions respectively. Now logic tells us that one of the two is the required function (this is the law of the excluded middle), unfortunately we do not know which function it is. So for single problems (i.e. problems without a parameter), it does not make sense in the framework of recursion theory to discuss decidability. As we have seen, intuitionistic logic sees this 'pathological example' in a different light.

(2) It is not decidable if $\{x\}$ is a total function.
Suppose it were decidable, then we would have a recursive function f such that $f(x) = 0 \Leftrightarrow \{x\}$ is total. Now consider

$$\varphi(x, y) := \begin{cases} 0 \text{ if } x \in K \\ \uparrow \text{ else} \end{cases}$$

By the S_n^m theorem there is a recursive h such that $\{h(x)\}(y) = \varphi(x, y)$. Now $\{h(x)\}$ is total $\Leftrightarrow x \in K$, so $f(h(x)) = 0 \Leftrightarrow x \in K$, i.e. we have a recursive characteristic function $\overline{\text{sg}}(f(h(x)))$ for K. Contradiction. Hence such an f does not exist, that is $\{x | \{x\} \text{ is total}\}$ is not recursive.

(3) The problem 'W_e is finite' is not recursively solvable.
In words, 'it is not decidable whether a recursively enumerable set is finite'.
Suppose that there was a recursive function f such that $f(e) = 0 \Leftrightarrow W_e$ is finite. Consider the $h(x)$ defined in example (2). Clearly $W_{h(x)} = \text{Dom}\{h(x)\} = \emptyset \Leftrightarrow x \notin K$, and $W_{h(x)}$ is infinite for $x \in K$. $f(h(x)) = 0 \Leftrightarrow x \notin K$, and hence $\text{sg}(f(h(x)))$ is a recursive characteristic function for K. Contradiction.

Note that $x \in K \Leftrightarrow \{x\}x \downarrow$, so we can reformulate the above solutions as follows: in (2) take $\varphi(x, y) = 0 \cdot \{x\}(x)$ and in (3) $\varphi(x, y) = \{x\}(x)$.

(4) The equality of RE sets is undecidable.
That is, $\{(x, y) | W_x = W_y\}$ is not recursive. We reduce the problem to the solution of (3) by choosing $W_y = \emptyset$.

(5) It is not decidable if W_e is recursive.
Put $\varphi(x, y) = \{x\}(x) \cdot \{y\}(y)$, then $\varphi(x, y) = \{h(x)\}(y)$ for a certain recursive h, and

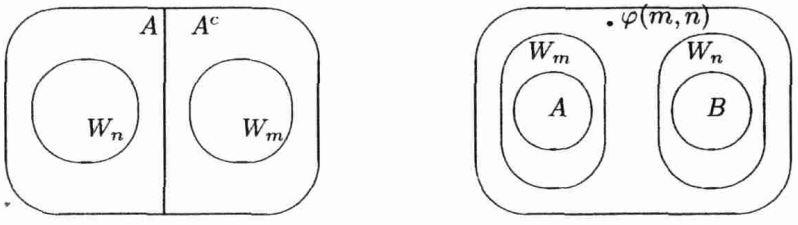
$$\text{Dom}\{h(x)\} = \begin{cases} K \text{ if } x \in K \\ \emptyset \text{ otherwise.} \end{cases}$$

Suppose there were a recursive function f such that $f(x) = 0 \Leftrightarrow W_x$ is recursive, then $f(h(x)) = 0 \Leftrightarrow x \notin K$ and, hence, K would be recursive. Contradiction.

There are several more techniques for establishing undecidability. We will treat here the method of inseparability.

Definition 7.3.8 *Two disjoint RE-sets W_m and W_n are* recursively separable *if there is a recursive set A such that $W_n \subseteq A$ and $W_m \subseteq A^c$. Disjoint sets*

A and B are effectively inseparable *if there is a partial recursive φ such that for every m, n with $A \subseteq W_m, B \subseteq W_n, W_m \cap W_n = \emptyset$ we have $\varphi(m,n) \downarrow$ and $\varphi(m,n) \notin W_m \cup W_n$.*



We immediately see that effectively inseparable RE sets are recursively inseparable, i.e. not recursively separable.

Theorem 7.3.9 *There exist effectively inseparable RE sets.*

Proof. Define $A = \{x|\{x\}(x) = 0\}, B = \{x|\{x\}(x) = 1\}$. Clearly $A \cap B = \emptyset$ and both sets are RE.

Let $W_m \cap W_n = \emptyset$ and $A \subseteq W_m, B \subset W_n$. To define φ we start testing $x \in W_m$ or $x \in W_n$; if we first find $x \in W_m$, we put an auxiliary function $\sigma(x)$ equal to 1, if x turns up first in W_n then we put $\sigma(x) = 0$.
Formally

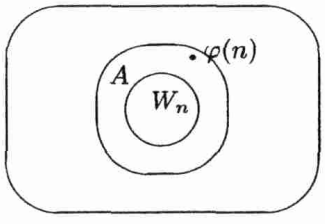
$$\sigma(m,n,x) = \begin{cases} 1 \text{ if } \exists z(T(m,x,z) \text{ and } \forall y < z \neg T(n,x,y)) \\ 0 \text{ if } \exists z(T(n,x,z) \text{ and } \forall y \leq z \neg T(m,x,y)) \\ \uparrow \text{ else.} \end{cases}$$

By the S_n^m theorem $\{h(m,n)\}(x) = \sigma(m,n,x)$ for some recursive h.

$$\begin{aligned} h(m,n) \in W_m &\Rightarrow h(m,n) \notin W_n. \text{ So } \exists z(T(m,h(m,n),z) \wedge \\ &\quad \forall y < z \neg T(n,h(m,n),y)) \\ &\Rightarrow \sigma(m,n,h(m,n)) = 1 \Rightarrow \{h(m,n)\}(h(m,n)) = 1 \\ &\Rightarrow h(m,n) \in B \Rightarrow h(m,n) \in W_n. \end{aligned}$$

Contradiction. Hence $h(m,n) \notin W_m$. Similarly $h(m,n) \notin W_n$. Thus h is the required φ. □

Definition 7.3.10 *A subset A of $\mathbb{N}$ is productive if there is a partial recursive function φ, such that for each $W_n \subseteq A$, $\varphi(n) \downarrow$ and $\varphi(n) \in A - W_n$.*



The theorem above gives us the following

Corollary 7.3.11 *There are productive sets.*

Proof. The set A^c defined in the above proof is *productive*. Let $W_k \subseteq A^c$. Put $W_\ell = B \cup W_k = W_n \cup W_k = W_{h(n,k)}$ for a suitable recursive function h. Now apply the separating function from the proof of the preceding theorem to $A = W_m$ and $W_{h(n,k))}$: $\varphi(m, h(n,k)) \in A^c - W_m$. □

Productive sets are in a strong sense not RE: no matter how one tries to fit an RE set into them, one can uniformly and effectively indicate a point that is missed by this RE set.

Exercises

1. The projection of an RE set is RE, i.e. if $R(\vec{x}, y)$ is RE then so is $\exists y R(\vec{x}, y)$.
2. (i) If A is enumerated by a strictly monotone function, then A is recursive. (ii) If A is infinite and recursive, then A is enumerated by a strictly increasing recursive function. (iii) An infinite RE set contains an infinite recursive subset.
3. Every non-empty RE set is enumerated by a total recursive function.
4. If A is an RE set and f a partially recursive function, then $f^{-1}(A)(= \{x | f(x) \in A\})$ and $f(A)$ are RE.
5. Show that the following are not recursive
   (i) $\{(x,y) | W_x = W_y\}$
   (ii) $\{x | W_x$ is recursive$\}$
   (iii) $\{x | 0 \in W_x\}$

7.4 Some arithmetic

In the section on recursive functions we have been working in the standard model of arithmetic; as we are now dealing with provability *in* arithmetic we have to avoid semantical arguments. and to rely solely on derivations inside the formal system of arithmetic. The generally accepted theory for arithmetic goes back to Peano, and thus we speak of *Peano arithmetic*, **PA** (cf. 2.7)

A major issue in the late 1920s was the completeness of **PA**. Gödel put an end to prevailing high hopes of the day by showing that **PA** is incomplete (1931). In order to carry out the necessary steps for Gödel's proof, we have to prove a number of theorems in **PA**. Most of these facts can be found in texts on number theory, or on the foundation of arithmetic. We will leave a considerable number of proofs to the reader. Most of the time one has to apply a suitable form of induction. Important as the actual proofs are, the heart of Gödel's argument lies in his ingenious incorporation of recursion theoretic arguments inside **PA**.

One of the obvious stumbling blocks for a straightforward imitation of 'self-reference' is the apparent poverty of the language of **PA**. It does not allow us to speak of, e.g., a finite string of numbers. Once we have exponentiation we can simply code finite sequences of numbers. Gödel showed that one can indeed

define the exponential (and much more) at the cost of some extra arithmetic, yielding his famous β-function. In 1971 Matiyashevich showed by other means that the exponential is definable in **PA**, thus enabling us to handle coding of sequences in **PA** directly. Peano arithmetic plus exponentiation is prima facie stronger than **PA**, but the above mentioned results show that exponentiation can be eliminated. Let us call the extended system **PA**; no confusion will arise.

We repeat the axioms:

- $\forall x(S(x) \neq 0)$,
- $\forall xy(S(x) = S(y) \to x = y)$,
- $\forall x(x + 0 = x)$,
- $\forall xy(x + S(y) = S(x + y))$,
- $\forall x(x \cdot 0 = 0)$,
- $\forall xy(x \cdot S(y) = x \cdot y + x)$,
- $\forall x(x^0 = 1)$,
- $\forall xy(x^{Sy} = x^y \cdot x)$,
- $\varphi(0) \wedge \forall x(\varphi(x) \to \varphi(S(x))) \to \forall x\varphi(x)$.

Since $\vdash \overline{1} = S(0)$, we will use both $S(x)$ and $x + \overline{1}$, whichever is convenient. We will also use the usual abbreviations. In order to simplify the notation, we will tacitly drop the '**PA**' in front of '$\vdash$' whenever possible. As another harmless simplification of notation we will often simply write n for $\overline{n}$ when no confusion arises.

In the following we will give a number of theorems of **PA**; in order to improve the readability, we will drop the universal quantifiers preceding the formulas. The reader should always think of 'the universal closure of ...'.

Furthermore we will use the standard abbreviations of algebra, i.e. leave out the multiplication dot, superfluous brackets, etc., when no confusion arises. We will also write 'n' instead of '$\overline{n}$'.

The basic operations satisfy the well-known laws:

Lemma 7.4.1 . *Addition and multiplication are associative and commutative, and $\cdot$ distributes over $+$.*

(i) $\vdash (x + y) + z = x + (y + z)$
(ii) $\vdash x + y = y + x$
(iii) $\vdash x(yz) = (xy)z$
(iv) $\vdash xy = yx$
(v) $\vdash x(y + z) = xy + xz$
(vi) $\vdash x^{y+z} = x^y x^z$
(vii) $\vdash (x^y)^z = x^{yz}$

Proof. Routine. □

Lemma 7.4.2 *(i)* $\vdash x = 0 \vee \exists y(x = Sy)$
(ii) $\vdash x + z = y + z \to x = y$
(iii) $\vdash z \neq 0 \to (xz = yz \to x = y)$
(iv) $\vdash x \neq 0 \to (x^y = x^z \to y = z)$
(v) $\vdash y \neq 0 \to (x^y = z^y \to x = z)$

Proof. Routine. □

A number of useful facts is listed in the exercises.

Although the language of **PA** is modest, many of the usual relations and functions can be defined. The ordering is an important example.

Definition 7.4.3 $x < y := \exists z(x + Sz = y)$

We will use the following abbreviations:

$x < y < z$	stands for	$x < y \wedge y < z$
$\forall x < y \varphi(x)$	,,	$\forall x(x < y \to \varphi(x)$
$\exists x < y \varphi(x)$	,,	$\exists x(x < y \wedge \varphi(x)$
$x > y$	,,	$y < x$
$x \leq y$	,,	$x < y \vee x = y.$

Theorem 7.4.4 *(i)* $\vdash \neg x < x$
(ii) $\vdash x < y \wedge y < z \to x < z$
(iii) $\vdash x < y \vee x = y \vee y < x$
(iv) $\vdash 0 = x \vee 0 < x$
(v) $\vdash x < y \to Sx = y \vee Sx < y$
(vi) $\vdash x < Sx$
(vii) $\vdash \neg x < y \wedge y < Sx$
(viii) $\vdash x < Sy \leftrightarrow (x = y \vee x < y)$
(ix) $\vdash x < y \leftrightarrow x + z < y + z$
(x) $\vdash z \neq 0 \to (x < y \leftrightarrow xz < yz)$
(xi) $\vdash x \neq 0 \to (0 < y < z \to x^y < x^z)$
(xii) $\vdash z \neq 0 \to (x < y \to x^z < y^z)$
(xiii) $\vdash x < y \leftrightarrow Sx < Sy$

Proof: routine. □

Quantification with an explicit bound can be replaced by a repeated disjunction or conjunction.

Theorem 7.4.5 $\vdash \forall x < \overline{n}\varphi(x) \leftrightarrow \varphi(0) \wedge \ldots \wedge \varphi(\overline{n-1}), (n > 0)$,
$\vdash \exists x < \overline{n}\varphi(x) \leftrightarrow \varphi(0) \vee \ldots \vee \varphi(\overline{n-1}), (n > 0)$

Proof. Induction on n. □

Theorem 7.4.6 *(i) well-founded induction*
$\vdash \forall x(\forall y < x\ \varphi(y) \to \varphi(x)) \to \forall x\varphi(x)$
(ii) least number principle (LNP)
$\vdash \exists x\varphi(x) \to \exists x(\varphi(x) \wedge \forall y < x \neg\varphi(y)))$

Proof: (i) Let us put $\psi(x) := \forall y < x\varphi(y)$. We assume $\forall x(\psi(x) \to \varphi(x))$ and proceed to apply induction on $\psi(x)$.
Clearly $\vdash \psi(0)$.
So let by induction hypothesis $\psi(x)$.
Now $\psi(Sx) \leftrightarrow [\forall y < Sx\varphi(y)] \leftrightarrow [\forall y((y = x \vee y < x) \to \varphi(y))] \leftrightarrow$
$[\forall y((y = x \to \varphi(y)) \wedge (y < x \to \varphi(y)))] \leftrightarrow [\forall y(\varphi(x) \wedge (y < x \to \varphi(y)))] \leftrightarrow$
$[\varphi(x) \wedge \forall y < x\varphi(y)] \leftrightarrow [\varphi(x) \wedge \psi(x)]$
Now $\psi(x)$ was given and $\psi(x) \to \varphi(x)$. Hence we get $\psi(Sx)$. This shows $\forall x\psi(x)$, and thus we derive $\forall x\varphi(x)$.
(ii) Consider the contraposition and reduce it to (i).

In our further considerations the following notions play a role.

Definition 7.4.7 *(i) Divisibility*
$x|y := \exists z(xz = y)$
(ii) Cut-off subtraction.
$z = y \dot{-} x := (x < y \wedge x + z = y) \vee (y \leq x \wedge z = 0)$
(iii) Remainder after division.
$z = rem(x, y) := (x \neq 0 \wedge \exists u(y = ux + z) \wedge z < x) \vee (x = 0 \wedge z = y)$
(iii) x is prime.
$Prime(x) := x > 1 \wedge \forall yz(x = yz \to y = x \vee y = 1)$

The right hand sides of (ii) and (iii) indeed determine functions, as shown in

Lemma 7.4.8 *(i)* $\vdash \forall xy\exists!z((x < y \wedge z + x = y) \vee (y \leq x \wedge z = 0))$
(ii) $\vdash \forall xy\exists!z((x \neq 0 \wedge \exists u(y = ux + z) \wedge z < y) \vee (x = 0 \wedge z = 0))$.

Proof. In both cases induction on y.

There is another characterization of the prime numbers.

Lemma 7.4.9 *(i)* $\vdash Prime(x) \leftrightarrow x > 1 \wedge \forall y(y|x \to y = 1 \vee y = x)$
(ii) $\vdash Prime(x) \leftrightarrow x > 1 \wedge \forall yz(x|yz \to x|y \vee x|z)$

Proof (i) is a mere reformulation of the definition.
(ii) $\to$ is a bit tricky. We introduce a bound on the product yz, and do wf-induction on the bound. Put $\varphi(w) = \forall yz \leq w(x|yz \to x|y \vee x|z)$. We now show $\forall w(\forall v < w\varphi(v) \to \varphi(w))$

Let $\forall v < w\varphi(v)$ and assume $\neg\varphi(w)$, i.e. there are $y, z \leq w$ such that $x|yz, \neg\, x|y, \neg\, x|z$ We will 'lower' the y such that the w is also lowered. Since

$\neg\, x|y, \neg\, x|z$, we have $z \neq 0$. Should $y \geq x$, then we may replace it by $y = rem(x, y)$ and carry on the argument. So let $y < x$. Now we once more get the remainder, $x = ay + b$ with $b < y$. We consider $b = 0$ and $b > 0$.

If $b = 0$, then $x = ay$; hence $y = 1 \vee y = x$. If $y = 1$, then $x|z$. Contradiction.

If $y = x$ then $x|y$. Contradiction.

Now if $b > 0$, then $bz = (x - ay)z = xz - ayz$. Since $x|yz$, we get $x|bz$. Observe that $bz < yz < w$, so we have a contradiction with $\forall v < w\varphi(v)$. Hence by *RAA* we have established the required statement.

For $\leftarrow$ we only have to apply the established facts about divisibility.

Can we prove in Peano's arithmetic that there are any primes? Yes, for example $PA \vdash \forall x(x > 1 \rightarrow \exists y(Prime(y) \wedge y|x)$

Proof. Observe that $\exists y(y > 1 \wedge y|x)$. By the LNP there is a least such y: $\exists y(y > 1 \wedge y|x \wedge \forall z < y(z > 1 \rightarrow \neg\, z|y))$
Now it is easy to show that this minimal y is a prime.

Primes and exponents are most useful for coding finite sequences of natural numbers, and hence for coding in general. There are many more codings, and some of them are more realistic in the sense that they have a lower complexity. For our purpose, however, primes and exponents will do.

As we have seen, we can code a finite sequence $(n_0 \ldots, n_{k-1})$ as the number $2^{n_0+1} \cdot 3^{n_1+1} \ldots p_{k-1}^{n_{k-1}+1}$

We will introduce some auxillary predicates.

Definition 7.4.10 (Successive primes) $Succprimes(x, y) := x < y \wedge Prime(x) \wedge Prime(y) \wedge \forall z(x < z < y \rightarrow \neg Prime(z))$

The next step is to define the sequence of prime numbers $2, 3, 5, \ldots p_n, \ldots,$. The basic trick here is that we consider all successive primes with ascending exponents: $2^0, 3^1, 5^2, 7^3, \ldots p_x^x$. We form the product and then pick the last factor.

Definition 7.4.11 (The xth prime number, p_x) $p_x = y := \exists z(\neg 2|z \wedge \forall v < y$
$\forall u \leq y(Succprime(v, u) \rightarrow \forall w < z(v^w|z \rightarrow u^{w+1}|z)) \wedge y^x|z \wedge \neg\, y^{x+1}|z)$

Observe that, as the definition yields a function, we have to show

Lemma 7.4.12 $\vdash \exists z(\neg 2|z \wedge \forall v < y_0 \forall u \leq y_0(Succprime(v, u) \rightarrow \forall w < z(v^w|z \rightarrow u^{w+1}|z)) \wedge y_0^x|z \wedge \neg\, y_0^{x+1}|z) \wedge$
$\exists z(\neg 2|z \wedge \forall v < y_1 \forall u \leq y_1(Succprime(v, u) \rightarrow \forall w < z(v^w|z \rightarrow u^{w+1}|z)) \wedge y_1^x|z \wedge \neg\, y_1^{x+1}|z) \rightarrow y_0 = y_1$

The above definition just mimicks the informal description. Note that we can bound the existential quantifier as $\exists z < y^{x^2}$ We have now justified the notation of sequence numbers as products of the form

$$p_0{}^{n_0+1} \cdot p_1{}^{n_1+1} \cdot \ldots \cdot p_{k-1}{}^{n_{k-1}+1}$$

The reader should check that according to the definition $p_0 = 2$. The decoding can also be defined. In general one can define the power of a prime factor.

Definition 7.4.13 (decoding) $(z)_k = v := p_k^{v+1}|z \wedge \neg p_k^{v+2}|z$

The length of a coded sequence can also be extracted from the code:

Definition 7.4.14 (length) $lth(z) = x := p_x|z \wedge \forall y < z(p_y|z \rightarrow y < x)$

Lemma 7.4.15 $\vdash Seq(z) \rightarrow (lth(z) = x \leftrightarrow (p_x|z \wedge \neg p_{x+1}|z))$

We define separately the coding of the empty sequence: $\langle\rangle := 1$
The coding of the sequence $(x_0, \ldots, x_{n-1})$ is denoted by $\langle x_0, \ldots, x_{n-1}\rangle$

Operations like concatenation and restriction of coded sequences can be defined such that

$\langle x_0 \ldots x_{n-1}\rangle * \langle y_0 \ldots y_{m-1}\rangle = \langle x_0, \ldots x_{n-1}, y_0 \ldots y_{m-1}\rangle$

$\langle x_0 \ldots x_n - 1\rangle|m = \langle x_0 \ldots x_m - 1\rangle$, where $m \leq n$ (warning: here | is used for the restriction relation, do not confuse with divisibility).

The tail of a sequence is defined as follows:

$$tail(y) = z \leftrightarrow (\exists x(y = \langle x\rangle * z) \vee (lth(y) = 0 \wedge z = 0)).$$

Closed terms of **PA** can be evaluated in **PA**:

Lemma 7.4.16 *For any closed term t there is a number n such that $\vdash t = \overline{n}$.*

Proof. External induction on t, cf. lemma 2.3.3. Observe that n is uniquely determined.

Corollary 7.4.17 $\mathbb{N} \models t_1 = t_2 \Rightarrow \vdash t_1 = t_2$ *for closed* t_1, t_2.

Gödel's theorem will show that in general '*true in the standard model*' (we will from now on just say 'true') and provable in **PA** are not the same. However for a class of simple sentences this is correct.

Definition 7.4.18 *(i) The class Δ_0 of formulas is inductively defined by*

$\varphi \in \Delta_0$ *for atomic* φ
$\varphi, \varphi \in \Delta_0 \Rightarrow \neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \rightarrow \psi \in \Delta_0$
$\varphi \in \Delta_0 \Rightarrow \forall x < y\varphi, \exists x < y\varphi \in \Delta_0$

(ii) The class Σ_1 is given by:

$\varphi, \neg\varphi \in \Sigma_1$ *for atomic* φ
$\varphi, \psi \in \Sigma_1 \Rightarrow \varphi \vee \psi, \varphi \wedge \psi \in \Sigma_1$
$\varphi \in \Sigma_1 \Rightarrow \forall x < y\varphi, \exists x < y\varphi, \exists x\varphi \in \Sigma_1$

A formula is called strict Σ_1 if it is of the form $\exists\vec{x}\varphi(\vec{x})$, where φ is Δ_0.

We will call formulas in the classes Δ_0 and Σ_1, Δ_0, Σ_1-formulas respectively. Formulas, provably equivalent to Σ_1 formulas, will also be called Σ_1 formulas.

For Σ_1-formulas we have that 'true = provable'.

Lemma 7.4.19 $\vdash \varphi$ *or* $\vdash \neg\varphi$, *for* Δ_0*-sentences* φ.

Proof. Induction on φ.
(i) φ atomic. If $\varphi \equiv t_1 = t_2$ and $t_1 = t_2$ is true, see corr. 7.4.17.
If $t_1 = t_2$ is false, then $t_1 = n$ and $t_2 = m$, where, say, $n = m + k$ with $k > 0$.
Now assume (in **PA**) $t_1 = t_2$, then $\overline{m} = \overline{m} + \overline{k}$. By 7.4.2 we get $0 = \overline{k}$, 7.4.19
But since $k = S(l)$ for some l, we obtain a contradiction. Hence $\vdash \neg t_1 = t_2$.
(ii) The induction cases are obvious. For $\forall x < t\varphi(x)$, where t is a closed term, use the identity $\forall x < \overline{n}\varphi(x) \leftrightarrow \varphi(0) \wedge \ldots \wedge \varphi(n-1)$. Similarly for $\exists x < t\varphi(x)$.

Theorem 7.4.20 (Σ_1-completeness) $\mathbb{N} \models \varphi \Leftrightarrow \mathbf{PA} \vdash \varphi$, *for* Σ_1 *sentences* φ.

Proof. Since the truth of $\exists x\varphi(x)$ comes to the truth of $\varphi(\overline{n})$ for some n, we may apply the above lemma.

7.5 Representability

In this section we will give the formalization of all this in **PA**, i.e. we will show that definable predicates exist corresponding with the predicates introduced above (in the standard model) – and that their properties are provable.

Definition 7.5.1 (representability) – *a formula* $\varphi(x_0, \ldots, x_{n-1}, y)$ *represents an n-ary function* f *if for all* $k_0, \ldots, k_{n-1}$

$$f(k_0, \ldots, k_{n-1}) = p \Rightarrow \vdash \forall y(\varphi(\bar{k}_0, \ldots, \bar{k}_{n-1}, y) \leftrightarrow y = \bar{p})$$

– *a formula* $\varphi(x_0, \ldots, x_{n-1})$ *represents a predicate* P *if for all* $k_0, \ldots, k_{n-1}$

$$P(k_0, \ldots, k_{n-1}) \Rightarrow \vdash \varphi(\bar{k}_0, \ldots, \bar{k}_{n-1})$$

and

$$\neg P(k_1, \ldots, k_n) \Rightarrow \vdash \neg\varphi(\bar{k}_0, \ldots, \bar{k}_{n-1})$$

– *a term* $t(x_0, \ldots, x_{n-1})$ *represents* f *if for all* $k_0, \ldots, k_{n-1}$

$$f(k_0, \ldots, k_{n-1}) = p \Rightarrow \vdash t(\bar{k}_0, \ldots, \bar{k}_{n-1}) = \bar{p}$$

Lemma 7.5.2 *If* f *is representable by a term, then* f *is representable by a formula.*

Proof: Let f be represented by t. Let $f(\mathbf{k}) = p$. Then $\vdash t(\bar{\mathbf{k}}) = \bar{p}$. Now define the formula $\varphi(\vec{x}, y) := t(\vec{x}) = y$. Then we have $\vdash \varphi(\bar{\mathbf{k}}, \bar{p})$. And hence $\bar{p} = y \to \varphi(\bar{k}, y)$. This proves $\vdash \varphi(\bar{\mathbf{k}}, y) \leftrightarrow \bar{p} = y$.

Sometimes it is convenient to split the representability of functions into two clauses.

Lemma 7.5.3 *A k-ary function is representable by φ iff*
$f(n_0 - n_{k-1}) = m \Rightarrow \vdash \varphi(\overline{n_0}, \ldots, \overline{n_{k-1}}, \bar{m})$ *and* $\vdash \exists! z \varphi(\overline{n_0} \ldots \overline{n_{k-1}}, z)$.

Proof. Immediate. Note that the last clause can be replaced by
$\vdash \varphi(\overline{n_0} \ldots \overline{n_{k-1}}, z) \to z = \bar{m}$.

The basic functions of arithmetic have their obvious representing terms. Quite simple functions can however not be represented by terms. E.g., the sigma function is represented by $\varphi(x, y) := (x = 0 \wedge y = 0) \vee (\neg x = 0 \wedge y = 1)$, but not by a term. However we can easily show $\vdash \forall x \exists! y \varphi(x, y)$, and therefore we could conservatively add the *sg* to **PA** (cf. 3.4.6). Note that quite a number of useful predicates and functions have Δ_0 formulas as a representation.

Lemma 7.5.4 *P is representable $\Leftrightarrow$ K_P is representable.*

Proof: Let $\varphi(\vec{x})$ represent P. Define $\psi(\vec{x}, y) = (\varphi(\vec{x}) \wedge (y = 1)) \vee (\neg\varphi(\vec{x}) \wedge (y = 0))$. Then ψ represents K_P, because if $K_P(\mathbf{k}) = 1$, then $P(\mathbf{k})$, so $\vdash \varphi(\bar{\mathbf{k}})$ and $\vdash \psi(\bar{\mathbf{k}}, y) \leftrightarrow (y = 1)$, and if $K_P(\mathbf{k}) = 0$, then $\neg P(\mathbf{k})$, so $\vdash \neg\varphi(\bar{\mathbf{k}})$ and $\vdash \psi(\bar{\mathbf{k}}, y) \leftrightarrow (y = 0)$. Conversely, let $\psi(\vec{x}, y)$ represent K_P. Define $\varphi(\vec{x}) := \psi(\vec{x}, 1)$. Then φ represents P,

There is a large class of representable functions, it includes the primitive recursive functions.

Theorem 7.5.5 *The primitive recursive functions are representable.*

Proof. Induction on the definition of primitive recursive function. It is simple to show that the initial functions are representable. The constant function C_m^k is represented by the term $\bar{m}$, the successor function S is represented by $x + 1$, and the the projection function P_i^k is represented by x_i.
The representable functions are closed under substitution and primitive recursion We will indicate the proof for the closure under primitive recursion.

$$\text{Consider } \begin{cases} f(\vec{x}, 0) & = g(\vec{x}) \\ f(\vec{x}, y+1) & = h(f(\vec{x}, y), \vec{x}, y) \end{cases}$$

g is represented by φ, h is represented by ψ :

$$g(\vec{n}) = m \Rightarrow \begin{cases} \vdash \varphi(\vec{n}, m) \text{ and} \\ \vdash \varphi(\vec{n}, y) \to y = m \end{cases}$$

$$h(p, \vec{n}, q) = m \Rightarrow \begin{cases} \vdash \psi(p, \vec{n}, q, m) \text{ and} \\ \vdash \psi(p, \vec{n}, q, y) \to y = m \end{cases}$$

Claim: f is represented by $\sigma(\vec{x}, y, z)$, which is mimicking $\exists w \in Seq(lth(w) = y + 1 \wedge ((w)_0 = g(\vec{x}) \wedge \forall i \leq y((w)_{i+1} = h((w)_i, \vec{x}, i) \wedge z = (w)_y)$

$$\begin{aligned}\sigma(\vec{x}, y, z) := \exists w \in Seq(lth(w) = y + 1 \wedge \varphi(\vec{x}, (w)_0) \wedge \\ \forall i \leq y(\psi((w)_i, \vec{x}, i, (w)_{i+1}) \wedge z = (w)_y)\end{aligned}$$

Now let $f(\vec{n}, p) = m$, then

$$\begin{cases} f(\vec{n}, 0) = g(\vec{n}) & = a_0 \\ f(\vec{n}, 1) = h(f(\vec{n}, 0), \vec{n}, 0) & = a_1 \\ f(\vec{n}, 2) = h(f(\vec{n}, 1), \vec{n}, 1) & = a_2 \\ \quad \ldots & \\ f(\vec{n}, p) = h(f(\vec{n}, p-1), \vec{n}, p-1) & = a_p = m \end{cases}$$

Put $w = \langle a_0, \ldots, a_p \rangle$; note that $lth(w) = p + 1$.

$$\begin{aligned} g(\vec{n}) = f(\vec{n}, 0) = a_0 &\Rightarrow\vdash \varphi(\vec{n}, a_0) \\ f(\vec{n}, 1) = a_1 &\Rightarrow\vdash \psi(a_0, \vec{n}, 0, a_1) \\ &\vdots \\ f(\vec{n}, p) = a_p &\Rightarrow\vdash \psi(a_{p-1}, \vec{n}, p-1, a_p) \end{aligned}$$

Therefore we have $\vdash lth(w) = p + 1 \wedge \varphi(\vec{n}, a_0) \wedge \psi(a_0, \vec{n}, 0, a_1) \wedge \ldots \wedge \psi(a_{p-1}, \vec{n}, p-1, a_p) \wedge (w)_p = m$ and hence $\vdash \sigma(\vec{n}, p, m)$.

Now we have to prove the second part: $\vdash \sigma(\vec{n}, p, z) \rightarrow z = m$. We prove this by induction on p.

(1) p=0. Observe that $\vdash \sigma(\vec{n}, 0, z) \leftrightarrow \varphi(\vec{n}, z)$, and since φ represents g, we get $\vdash \varphi(\vec{n}, z) \rightarrow z = \overline{m}$

(2) $p = q + 1$. Induction hypothesis: $\vdash \sigma(\vec{n}, q, z) \rightarrow z = \overline{f(\vec{n}, q)} (= \overline{m})$

$\sigma(\vec{n}, q + 1, z) = \exists w \in Seq(lth(w) = q + 2 \wedge \varphi(\vec{n}, (w)_0) \wedge$

$\forall i \leq y(\psi((w)_i, \vec{n}, i, (w)_{i+1}) \wedge z = (w)_{q+1})$.

We now see that

$\vdash \sigma(\vec{n}, q + 1, z) \rightarrow \exists u(\sigma(\vec{n}, q, u) \wedge \psi(u, \vec{n}, q, z)$.

Using the induction hypothesis we get

$\vdash \sigma(\vec{n}, q + 1, z) \rightarrow \exists u(u = f(\vec{n}, q) \wedge \psi(u, \vec{n}, q, z))$

And hence $\vdash \sigma(\vec{n}, q + 1, z) \rightarrow \psi(f(\vec{n}, q), \vec{n}, q, z)$

Thus by the property of ψ: $\vdash \sigma(\vec{n}, q + 1, z) \rightarrow z = f(\vec{n}, q + 1)$

It is now one more step to show that all recursive functions are representable, for we have seen that all recursive functions can be obtained by a single minimalization from a primitive recursive predicate.

Theorem 7.5.6 *All recursive functions are representable.*

Proof. We show that the representable functions are closed under minimalization. Since representability for predicates is equivalent to representability for

functions, we consider the case $f(\vec{x}) = \mu y P(\vec{x}, y)$ for a predicate P represented by φ, where $\forall \vec{x} \exists y P(\vec{x}, y)$.
Claim: $\psi(\vec{x}, y) := \varphi(\vec{x}, y) \wedge \forall z < y \neg\varphi(\vec{x}, z)$ represents $\mu y P(\vec{x}, y)$.

$$\begin{aligned} m = \mu y P(\vec{n}, y) &\Rightarrow P(\vec{n}, m) \wedge \neg P(\vec{n}, 0) \wedge \ldots \wedge \neg P(\vec{n}, m-1) \\ &\Rightarrow \vdash \varphi(\vec{n}, m) \wedge \neg\varphi(\vec{n}, 0) \wedge \ldots \wedge \neg\varphi(\vec{n}, m-1) \\ &\Rightarrow \vdash \varphi(\vec{n}, m) \wedge \forall z < m \neg\varphi(\vec{n}, z) \\ &\Rightarrow \vdash \psi(\vec{n}, m) \end{aligned}$$

Now let $\varphi(\vec{n}, y)$ be given, then we have $\varphi(\vec{n}, y) \wedge \forall z < y \neg\varphi(\vec{n}, z)$. This immediately yields $m \geq y$. Conversely, since $\varphi(\vec{n}, m)$, we see that $m \leq y$. Hence $y = m$. This informal argument is straightforwardly formalized as $\vdash \varphi(\vec{n}, y) \rightarrow y = m$.

We have established that recursive sets are representable. One might perhaps hope that this can be extended to recursively enumerable sets. This happens not to be the case. We will consider the RE sets now.

Definition 7.5.7 *$R(\vec{x})$ is semi-representable in T if $R(\vec{n}) \Leftrightarrow T \vdash \varphi(\overline{\vec{n}})$ for a $\varphi(\vec{x})$.*

Theorem 7.5.8 *R is semi-representable $\Leftrightarrow$ R is recursively enumerable.*

For the proof see p. 251.

Corollary 7.5.9 *R is representable $\Leftrightarrow$ R is recursive.*

Exercises

1. Show
   $\vdash x + y = 0 \rightarrow x = 0 \wedge y = 0$
   $\vdash xy = 0 \rightarrow x = 0 \vee y = 0$
   $\vdash xy = 1 \rightarrow x = 1 \wedge y = 1$
   $\vdash x^y = 1 \rightarrow y = 0 \vee x = 1$
   $\vdash x^y = 0 \rightarrow x = 0 \wedge y \neq 0$
   $\vdash x + y = 1 \rightarrow (x = 0 \wedge y = 1) \vee (x = 1 \wedge y = 0)$
2. Show that all Σ_1-formulas are equivalent to prenex formulas with the existential quantifiers preceding the bounded universal ones (Hint. Consider the combination $\forall x < t \exists y \varphi(x, y)$, this yields a coded sequence z such that $\forall x < t \varphi(x, (z)_x)$). I.e. in **PA** Σ_1 formulas are equivalent to strict Σ_1 formulas.
3. Show that one can contract similar quantifiers. E.g. $\forall x \forall y \varphi(x, y) \leftrightarrow \forall z \varphi((z)_0, (z)_1)$.

7.6 Derivability

In this section we define a coding for a recursively enumerable predicate $Thm(x)$, that says "x is a theorem". Because of the minimization and upper bounds on quantifiers, all predicates and functions defined along the way are primitive recursive. Observe that we are back in recursion theory, that is in informal arithmetic.

Coding of the syntax

The function $\ulcorner - \urcorner$ codes the syntax. For the alphabet, it is given by

$\wedge$	$\rightarrow$	$\forall$	0	S	+	$\cdot$	exp	=	(	)	x_i
2	3	5	7	11	13	17	19	23	29	31	p_{11+i}

Next we code the terms.

$$\ulcorner f(t_1,\ldots,t_n)\urcorner \quad := \quad \langle \ulcorner f\urcorner, \ulcorner(\urcorner, \ulcorner t_1\urcorner, \ldots, \ulcorner t_n\urcorner, \ulcorner)\urcorner\rangle$$

Finally we code the formulas. Note that $\{\wedge,\rightarrow,\forall\}$ is a functionally complete set, so the remaining connectives can be defined.

$$\begin{aligned}
\ulcorner(t=s)\urcorner &:= \langle \ulcorner(\urcorner, \ulcorner t\urcorner, \ulcorner=\urcorner, \ulcorner s\urcorner, \ulcorner)\urcorner\rangle \\
\ulcorner(\varphi\wedge\psi)\urcorner &:= \langle \ulcorner(\urcorner, \ulcorner\varphi\urcorner, \ulcorner\wedge\urcorner, \ulcorner\psi\urcorner, \ulcorner)\urcorner\rangle \\
\ulcorner(\varphi\rightarrow\psi)\urcorner &:= \langle \ulcorner(\urcorner, \ulcorner\varphi\urcorner, \ulcorner\rightarrow\urcorner, \ulcorner\psi\urcorner, \ulcorner)\urcorner\rangle \\
\ulcorner(\forall x_i\varphi)\urcorner &:= \langle \ulcorner(\urcorner, \ulcorner\forall\urcorner, \ulcorner x_i\urcorner, \ulcorner\varphi\urcorner, \ulcorner)\urcorner\rangle
\end{aligned}$$

$Const(x)$ and $Var(x)$ characterize the codes of constants and variables, respectively.

$$\begin{aligned}
Const(x) &:= x = \ulcorner 0\urcorner \\
Var(x) &:= \exists i \leq x(p_{11+i} = x)
\end{aligned}$$

$$\begin{aligned}
Fnc1(x) &:= x = \ulcorner S\urcorner \\
Fnc2(x) &:= x = \ulcorner +\urcorner \vee x = \ulcorner\cdot\urcorner \vee x = \ulcorner exp\urcorner
\end{aligned}$$

$Term(x)$ — *x is a term* — and $Form(x)$ — *x is a formula* — are primitive recursive predicates according to the primitive recursive version of the recursion-theorem. Note that we will code according to the standard function notation, e.g. $+(x,y)$ instead of $x+y$.

$$\begin{aligned}
Term(x) := {} & Const(x) \vee Var(x) \vee \\
& \Big(Seq(x) \wedge lth(x) = 4 \wedge Fnc1((x)_0) \wedge \\
& (x)_1 = \ulcorner(\urcorner \wedge Term((x)_2) \wedge (x)_3 = \ulcorner)\urcorner\Big) \vee \\
& \Big(Seq(x) \wedge lth(x) = 5 \wedge Fnc2((x)_0) \wedge \\
& (x)_1 = \ulcorner(\urcorner \wedge Term((x)_2) \ \wedge \ Term((x)_3) \wedge (x)_4 = \ulcorner)\urcorner\Big)
\end{aligned}$$

$$\begin{aligned}Form(x) := Seq(x) \wedge lth(x) = 5 \wedge (x)_0 = \ulcorner(\urcorner \wedge (x)_4 = \ulcorner)\urcorner \wedge \\ \Big[\Big(Term((x)_1) \wedge (x)_2 = \ulcorner=\urcorner \wedge Term((x)_3)\Big) \vee \\ \Big(Form((x)_1) \wedge (x)_2 = \ulcorner\wedge\urcorner \wedge Form((x)_3)\Big) \vee \\ \Big(Form((x)_1) \wedge (x)_2 = \ulcorner\rightarrow\urcorner \wedge Form((x)_3)\Big) \vee \\ \Big((x)_1 = \ulcorner\forall\urcorner \wedge Var((x)_2) \wedge Form((x)_3)\Big)\Big]\end{aligned}$$

All kinds of syntactical notions can be coded in primitive recursive predicates, for example $Free(x,y)$ — *x is a free variable in y*, and $FreeFor(x,y,z)$ — *x is free for y in z*.

$$Free(x,y) := \begin{cases} \Big(Var(x) \wedge Term(y) \wedge \neg Const(y) \wedge \\ (Var(y) \rightarrow x = y) \wedge \\ (Fnc1((y)_0) \rightarrow Free(x,(y)_2)) \wedge \\ (Fnc2((y)_0) \rightarrow (Free(x,(y)_2) \vee Free(x,(y)_3)))\Big) \\ \qquad\qquad or \\ \Big(Var(x) \wedge Form(y) \wedge \\ ((y)_1 \neq \ulcorner\forall\urcorner \rightarrow (Free(x,(y)_1) \vee Free(x,(y)_3))) \wedge \\ ((y)_1 = \ulcorner\forall\urcorner \rightarrow (x \neq (y)_2 \wedge Free(x,(y)_4)))\Big) \end{cases}$$

$$FreeFor(x,y,z) := \begin{cases} Term(x) \wedge Var(y) \wedge Form(z) \wedge \\ \Big[\Big((z)_2 = \ulcorner=\urcorner\Big) \vee \\ \Big((z)_1 \neq \ulcorner\forall\urcorner \wedge FreeFor(x,y,(z)_1) \wedge FreeFor(x,y,(z)_3\Big) \vee \\ \Big((z)_1 = \ulcorner\forall\urcorner \wedge \neg Free((z)_2,x) \wedge \\ (Free(y,z) \rightarrow (Free((z)_2,x) \wedge Free(x,y,(z)_3)))\Big)\Big] \end{cases}$$

Having coded these predicates, we can define a substitution-operator Sub such that $Sub(\ulcorner\varphi\urcorner, \ulcorner x\urcorner, \ulcorner t\urcorner) = \ulcorner\varphi[t/x]\urcorner$.

$$Sub(x,y,z) := \begin{cases} x & \text{if } Const(x) \\ x & \text{if } Var(x) \wedge x \neq y \\ z & \text{if } Var(x) \wedge x = y \\ \langle (x)_0, \ulcorner(\urcorner, Sub((x)_2, y, z), \ulcorner)\urcorner \rangle & \text{if } Term(x) \wedge \\ & \quad Fnc1((x)_0) \\ \langle (x)_0, \ulcorner(\urcorner, Sub((x)_2, y, z), Sub((x)_3, y, z), \ulcorner)\urcorner \rangle & \text{if } Term(x) \wedge \\ & \quad Fnc2((x)_0) \\ \langle \ulcorner(\urcorner, Sub((x)_1, y, z), (x)_2, Sub((x)_3, y, z), \ulcorner)\urcorner \rangle & \text{if } Form(x) \wedge \\ & \quad FreeFor(x,y,z) \wedge \\ & \quad (x)_0 \neq \ulcorner\forall\urcorner \\ \langle \ulcorner(\urcorner, (x)_1, (x)_2, Sub((x)_3, y, z), \ulcorner)\urcorner \rangle & \text{if } Form(x) \wedge \\ & \quad FreeFor(z,y,x) \wedge \\ & \quad (x)_0 = \ulcorner\forall\urcorner \\ 0 & else \end{cases}$$

Clearly *Sub* is primitive recursive (course of value recursion).

Coding of derivability

Our next step is to obtain a primitive recursive predicate *Der* that says that x is a derivation with hypotheses $y_0, \ldots, y_{lth(y)-1}$ and conclusion z. Before that we give a coding of derivations.

initial derivation

$$[\varphi] = \langle 0, \varphi \rangle$$

$\wedge$ I

$$\left[\begin{array}{cc} D_1 & D_2 \\ \varphi & \psi \\ \hline (\varphi \wedge \psi) \end{array}\right] = \langle \langle 0, \ulcorner\wedge\urcorner \rangle, \begin{bmatrix} D_1 \\ \varphi \end{bmatrix}, \begin{bmatrix} D_2 \\ \psi \end{bmatrix}, \ulcorner(\varphi \wedge \psi)\urcorner \rangle$$

$\wedge$ E

$$\left[\begin{array}{c} D \\ \hline (\varphi \wedge \psi) \\ \hline \varphi \end{array}\right] = \langle \langle 1, \ulcorner\wedge\urcorner \rangle, \begin{bmatrix} D \\ (\varphi \wedge \psi) \end{bmatrix}, \ulcorner\varphi\urcorner \rangle$$

$\rightarrow$ I

$$\left[\begin{array}{c} \varphi \\ D \\ \psi \\ \hline (\varphi \rightarrow \psi) \end{array}\right] = \langle \langle 0, \ulcorner\rightarrow\urcorner \rangle, \begin{bmatrix} D \\ \psi \end{bmatrix}, \ulcorner(\varphi \rightarrow \psi)\urcorner \rangle$$

$\rightarrow$ E

$$\left[\begin{array}{cc} D_1 & D_2 \\ \varphi & (\varphi \rightarrow \psi) \\ \hline \psi \end{array}\right] = \langle \langle 1, \ulcorner\rightarrow\urcorner \rangle, \begin{bmatrix} D_1 \\ \varphi \end{bmatrix}, \begin{bmatrix} D_2 \\ (\varphi \rightarrow \psi) \end{bmatrix}, \ulcorner\psi\urcorner \rangle$$

RAA

$$\left[\begin{array}{c} (\varphi \rightarrow \bot) \\ D \\ \bot \\ \hline \varphi \end{array}\right] = \langle \langle 1, \ulcorner\bot\urcorner \rangle, \begin{bmatrix} D \\ \bot \end{bmatrix}, \ulcorner\varphi\urcorner \rangle$$

$\forall$ I

$$\left[\begin{array}{c} D \\ \varphi \\ \hline (\forall x\varphi) \end{array}\right] \quad = \quad \langle\langle 0, \ulcorner\forall\urcorner\rangle, \left[\begin{array}{c} D \\ \varphi \end{array}\right], \ulcorner(\forall x\varphi)\urcorner\rangle$$

$\forall$ E

$$\left[\begin{array}{c} D \\ (\forall x\varphi) \\ \hline \varphi[t/x] \end{array}\right] \quad = \quad \langle\langle 1, \ulcorner\forall\urcorner\rangle, \left[\begin{array}{c} D \\ (\forall x\varphi) \end{array}\right], \ulcorner\varphi[t/x]\urcorner\rangle$$

For *Der* we need a device to cancel hypotheses from a derivation. We consider a sequence y of (codes of) hypotheses and successively delete items u.

$$Cancel(u,y) := \begin{cases} y \text{ if } lth(y) = 0 \\ Cancel(u, tail(y)) \text{ if } (y)_0 = u \\ \langle (y)_0, Cancel(u, tail(y))\rangle \text{ if } (y)_0 \neq u \end{cases}$$

Here $tail(y) = z \Leftrightarrow (lth(y) > 0 \wedge \exists x(y = \langle x\rangle * z) \vee (lth(y) = 0 \wedge z = 0)$

Now we can code *Der*, where $Der(x,y,z)$ stands for 'x is the code of a derivation of a formula with code z from a coded sequence of hypotheses y'. In the definition of *Der* $\perp$ is defined as (0=1).

$$\begin{aligned} Der(x,y,z) := \quad & Form(z) \wedge \textstyle\bigwedge_{i=0}^{lth(y)-1} Form((i)_v) \wedge \\ & \Big[\Big(\exists i < lth(y)(z = (y)_i \wedge x = \langle 0, z\rangle\Big) \\ & \qquad \text{or} \\ & \Big(\exists x_1 x_2 \leq x \exists y_1 y_2 \leq x \exists z_1 z_2 \leq x y = y_1 * y_2 \wedge \\ & Der(x_1, y_1, z_1) \wedge Der(x_2, y_2, z_2) \wedge \\ & z = \langle \ulcorner(\urcorner, z_1, \ulcorner\wedge\urcorner, z_2, \ulcorner)\urcorner\rangle \wedge x = \langle\langle 0, \ulcorner\wedge\urcorner\rangle, x_1, x_2, z\rangle\Big) \\ & \qquad \text{or} \\ & \Big(\exists u \leq x \exists x_1 \leq x \exists z_1 \leq x\, Der(x_1, y, z_1) \wedge \\ & (z_1 = \langle \ulcorner(\urcorner, z, \ulcorner\wedge\urcorner, u, \ulcorner)\urcorner\rangle \vee (z_1 = \langle \ulcorner(\urcorner, u, \ulcorner\wedge\urcorner, z, \ulcorner)\urcorner\rangle) \wedge \\ & x = \langle\langle 1, \ulcorner\wedge\urcorner\rangle, x_1, z\rangle\Big) \\ & \qquad \text{or} \\ & \Big(\exists x_1 \leq x \exists y_1 \leq x \exists u \leq x \exists z_1 \leq x (y = Cancel(u, y_1) \vee \\ & y = y_1) \wedge Der(x_1, y_1, z_1) \wedge z = \langle \ulcorner(\urcorner, u, \ulcorner\rightarrow\urcorner, z_1, \ulcorner)\urcorner\rangle \wedge \\ & x = \langle\langle 0, \ulcorner\rightarrow\urcorner\rangle, x_1, z_1\rangle\Big) \\ & \qquad \text{or} \\ & \Big(\exists x_1 x_2 \leq x \exists y_1 y_2 \leq x \exists z_1 z_2 \leq x\ (y = y_1 * y_2 \wedge \\ & Der(x_1, y_1, z_1) \wedge Der(x_2, y_2, z_2) \wedge \\ & z_2 = \langle \ulcorner(\urcorner, z_1, \ulcorner\rightarrow\urcorner, z, \ulcorner)\urcorner\rangle \wedge x = \langle\langle 1, \ulcorner\rightarrow\urcorner\rangle, x_1, x_2, z\rangle)\Big) \\ & \qquad \text{or} \end{aligned}$$

$$\Big(\exists x_1 \le x \exists z_1 \le x \exists v \le x (Der(x_1, y, z_1) \wedge Var(v) \wedge$$
$$\textstyle\bigwedge_{i=0}^{lth(y)-1} \neg Free(v, (y)_i) \wedge z = \langle \ulcorner\forall\urcorner, v, \ulcorner(\urcorner, z_1, \ulcorner)\urcorner\rangle \wedge$$
$$x = \langle\langle 0, \ulcorner\forall\urcorner\rangle, x_1, z_1\rangle)\Big)$$
or
$$\Big(\exists t \le x \exists v \le x \exists x_1 \le x \exists z_1 \le x (Var(v) \wedge Term(t) \wedge$$
$$Freefor(t, v, z_1) \wedge z = Sub(z_1, v, t) \wedge$$
$$Der(x_1, y, \langle \ulcorner\forall\urcorner, v, \ulcorner(\urcorner, z_1, \ulcorner)\urcorner\rangle \wedge x = \langle\langle 1, \ulcorner\forall\urcorner\rangle, y, z\rangle)\Big)$$
or
$$\Big(\exists x_1 \le x \exists y_1 \le x \exists z_1 \le x (Der(x_1, y_1, \langle\ulcorner\bot\urcorner\rangle) \wedge$$
$$y = Cancel(\langle z, \ulcorner\to\urcorner, \ulcorner\bot\urcorner\rangle, y_1) \wedge x = \langle\langle 1, \ulcorner\bot\urcorner\rangle, x_1, z_1\rangle)\Big)\Big]$$

Coding of provability

The axioms of Peano's arithmetic are listed on page 238. However, for the purpose of coding derivability we have to be precise; we must include the axioms for identity. They are the usual ones (see 2.6 and 2.10.2), including the 'congruence axioms' for the operations:

$$(x_1 = y_1 \wedge x_2 = y_2) \to \Big(S(x_1) = S(y_1) \ \wedge \ x_1 + x_2 = y_1 + y_2 \ \wedge$$
$$x_1 \cdot x_2 = y_1 \cdot y_2 \ \wedge \ x_1^{x_2} = y_1^{y_2}\Big)$$

These axioms can easily be coded and put together in a primitive recursive predicate $Ax(x)$ — *x is an axiom.* The provability predicate $Prov(x, z)$ - x is a derivation of z from the axioms of $\mathcal{PA}$ - follows immediately.

$$Prov(x, z) \quad := \quad \exists y \le x \big(Der(x, y, z) \wedge \bigwedge_{i=0}^{lth(y)-1} Ax((y)_i)\big)$$

Finally we can define $Thm(x)$ — *x is a theorem.* Thm is recursively enumerable.

$$Thm(z) \quad := \quad \exists x Prov(x, z)$$

Having at our disposition the provability predicate, which is Σ_1^0, we can finish the proof of 'semi-representable =- RE' (Theorem 7.4.9).

Proof. For convenience let R be unary recursively enumerable set.
$\Rightarrow$: R is semi-representable by φ. $R(n) \Leftrightarrow \ \vdash \varphi(\overline{n}) \Leftrightarrow \ \exists y Prov\left(\ulcorner\varphi(\overline{n})\urcorner, y\right)$. Note that $\ulcorner\varphi(\overline{n})\urcorner$ is a recursive function of n. $Prov$ is primitive recursive, so R is recursively enumerable.
$\Leftarrow$: R is recursively enumerable $\Rightarrow \ R(n) = \exists x P(n, x)$ for a primitive recursive P. $P(n, m) \Leftrightarrow \ \vdash \varphi(\overline{n}, \overline{m})$ for some φ. $R(n) \Leftrightarrow \ P(n, m)$ for some $m \Leftrightarrow \ \vdash \varphi(\overline{n}, \overline{m})$ for some $m \Rightarrow \ \vdash \exists y \varphi(\overline{n}, y)$. Therefore we also have $\vdash \exists y \varphi(\overline{n}, y) \Rightarrow \ R(n)$. So $\exists y \varphi(\overline{n}, y)$ semi-represents R. □

7.7 Incompleteness

Theorem 7.7.1 (Fixpoint theorem) *For each formula $\varphi(x)$ (with $FV(\varphi) = \{x\}$) there exists a sentence ψ such that $\vdash \varphi(\overline{\ulcorner\psi\urcorner}) \leftrightarrow \psi$.*

Proof. Popular version: consider a simplified substitution function $s(x,y)$ which is the old substitution function for a fixed variable: $s(x,y) = Sub(x, \ulcorner x_0 \urcorner, y)$. Then define $\theta(x) := \varphi\big(s(x,x)\big)$. Let $m := \ulcorner\theta(x)\urcorner$, then put $\psi := \theta(\overline{m})$. Note that $\psi \leftrightarrow \theta(\overline{m}) \leftrightarrow \varphi\big(\overline{s(m,m)}\big) \leftrightarrow \varphi\big(\overline{s(\overline{\ulcorner\theta(x)\urcorner}, m)}\big) \leftrightarrow \varphi\big(\overline{\ulcorner\theta(\overline{m})\urcorner}\big) \leftrightarrow \varphi\big(\overline{\ulcorner\psi\urcorner}\big)$.

This argument would work if there were a function (or term) for s in the language. This could be done by extending the language with sufficiently many functions ("all primitive recursive functions" surely will do). Now we have to use representing formulas.
Formal version: let $\sigma(x,y,z)$ represent the primitive recursive function $s(x,y)$. Now suppose $\theta(x) := \exists y\big(\varphi(y) \wedge \sigma(x,x,y)\big), m = \ulcorner\theta(x)\urcorner$ and $\psi = \theta(\overline{m})$. Then

$$\psi \leftrightarrow \theta(\overline{m}) \leftrightarrow \exists y\big(\varphi(y) \wedge \sigma(\overline{m},\overline{m},y)\big) \tag{7.1}$$

$$\vdash \forall y\big(\sigma(\overline{m},\overline{m},y) \leftrightarrow y = \overline{s(m,m)}\big)$$

$$\vdash \forall y\big(\sigma(\overline{m},\overline{m},y) \leftrightarrow y = \overline{\ulcorner\theta(\overline{m})\urcorner}\big) \tag{7.2}$$

By logic (1) and (2) give $\psi \leftrightarrow \exists y\big(\varphi(y) \wedge y = \overline{\ulcorner\theta(\overline{m})\urcorner}\big)$ so $\psi \leftrightarrow \varphi\big(\overline{\ulcorner\theta(\overline{m})\urcorner}\big) \leftrightarrow \varphi\big(\overline{\ulcorner\psi\urcorner}\big)$. □

Definition 7.7.2 *(i)* **PA** *(or any other theory T of arithmetic) is called ω-complete if $\vdash \exists x\varphi(x) \Rightarrow \ \vdash \varphi(\overline{n})$ for some $n \in \mathbb{N}$.*
(ii) T is ω-consistent if there is no φ such that ($\vdash \exists x\varphi(x)$ and $\vdash \neg\varphi(\overline{n})$ for all n) for all φ.

Theorem 7.7.3 *(Gödel's first incompleteness theorem)*
If **PA** *is ω-consistent then* **PA** *is incomplete.*

Proof. Consider the predicate $Prov(x,y)$ represented by the formula $\overline{Prov}(x,y)$. Let $\overline{Thm}(x) := \exists y\overline{Prov}(x,y)$. Apply the fixpoint theorem to $\neg\overline{Thm}(x)$: there exists a φ such that $\vdash \varphi \leftrightarrow \neg\overline{Thm}(\overline{\ulcorner\varphi\urcorner})$. φ, the so-called *Gödel sentence*, says in **PA**: "I am not provable."

Claim 1: If $\vdash \varphi$ then **PA** is inconsistent.

Proof. $\vdash \varphi \Rightarrow$ there is a n such that $Prov(\ulcorner\varphi\urcorner, n)$, hence $\vdash \overline{Prov}(\overline{\ulcorner\varphi\urcorner}, \overline{n}) \Rightarrow$ $\vdash \exists y\overline{Prov}(\overline{\ulcorner\varphi\urcorner}, y) \Rightarrow \ \vdash \overline{Thm}(\overline{\ulcorner\varphi\urcorner}) \Rightarrow \ \vdash \neg\varphi$. Thus **PA** is inconsistent. □

Claim 2: If $\vdash \neg\varphi$ then **PA** is ω-inconsistent.

Proof. $\vdash \neg\varphi \Rightarrow \quad \vdash \overline{Thm}(\overline{\ulcorner\varphi\urcorner}) \Rightarrow \quad \vdash \exists x\overline{Prov}(\overline{\ulcorner\varphi\urcorner}, x)$. Suppose **PA** is ω-consistent; since ω-consistency implies consistency, we have $\nvdash \varphi$ and therefore $\neg Prov(\ulcorner\varphi\urcorner, n)$ for all n. Hence $\vdash \neg\overline{Prov}(\overline{\ulcorner\varphi\urcorner}, \overline{n})$ for all n. Contradiction. □

Remarks. In the foregoing we made use of the representability of the provability-predicate, which in turn depended on the representability of all recursive functions and predicates.

For the representability of $Prov(x, y)$, the set of axioms has to be recursively enumerable. So Gödel's first incompleteness theorem holds for all recursively enumerable theories in which the recursive functions are representable. So one cannot make **PA** complete by adding the Gödel sentence, the result would again be incomplete.

In the standard model $\mathbb{N}$ either φ, or $\neg\varphi$ is true. The definition enables us to determine which one. Notice that the axioms of **PA** are true in $\mathbb{N}$, so $\vDash \varphi \leftrightarrow \neg\overline{Thm}(\overline{\ulcorner\varphi\urcorner})$. Suppose $\mathbb{N} \vDash \overline{Thm}(\overline{\ulcorner\varphi\urcorner})$ then $\mathbb{N} \vDash \exists x\overline{Prov}(\overline{\ulcorner\varphi\urcorner}, x) \Leftrightarrow \mathbb{N} \vDash \overline{Prov}(\overline{\ulcorner\varphi\urcorner}, \overline{n})$ for some $n \Leftrightarrow \vdash \overline{Prov}(\ulcorner\varphi\urcorner, \overline{n})$ for some $n \Leftrightarrow \vdash \varphi \Rightarrow$ $\vdash \neg\overline{Thm}(\overline{\ulcorner\varphi\urcorner}) \Rightarrow \mathbb{N} \vDash \neg\overline{Thm}(\overline{\ulcorner\varphi\urcorner})$. Contradiction. Thus φ is true in $\mathbb{N}$. This is usually expressed as 'there is a true statement of arithmetic which is not provable'.

Remarks. It is generally accepted that **PA** is a *true* theory, that is $\mathbb{N}$ is a model of **PA**, and so the conditions on Gödel's theorem seem to be superfluous. However, the fact that **PA** is a true theory is based on a semantical argument. The refinement consists in considering arbitrary theories, without the use of semantics.

The incompleteness theorem can be freed of the ω-consistency-condition. We introduce for that purpose Rosser's predicate:

$$Ros(x) := \exists y\big(Prov(neg(x), y) \wedge \forall z < y\ \neg Prov(x, z)\big),$$

with $neg(\ulcorner\varphi\urcorner) = \ulcorner\neg\varphi\urcorner$. The predicate following the quantifier is represented by $\overline{Prov}(\overline{neg}(x), y) \wedge \forall z < y\ \neg\overline{Prov}(x, z)$. An application of the fixpoint theorem yields a ψ such that

$$\vdash \psi \leftrightarrow \exists y\big[\ \overline{Prov}(\overline{\ulcorner\neg\psi\urcorner}, y) \wedge \forall z < y\ \neg\overline{Prov}(\overline{\ulcorner\psi\urcorner}, z)\ \big] \qquad (1)$$

Claim: **PA** is consistent $\Rightarrow \nvdash \psi$ and $\nvdash \neg\psi$

Proof. (i) Suppose $\vdash \psi$ then there exists a n such that $Prov(\ulcorner\psi\urcorner, n)$ so $\vdash \overline{Prov}(\overline{\ulcorner\psi\urcorner}, \overline{n})$ (2)

From (1) and (2) it follows that $\vdash \exists y < \overline{n}\overline{Prov}(\overline{\ulcorner\neg\psi\urcorner}, y)$, i.e. $\vdash \overline{Prov}(\overline{\ulcorner\neg\psi\urcorner}, \overline{0}) \vee \ldots \vee \overline{Prov}(\overline{\ulcorner\neg\psi\urcorner}, \overline{n-1})$. Note that $\overline{Prov}$ is Δ_0, thus the following holds: $\vdash \sigma \vee \tau \Leftrightarrow \vdash \sigma$ or $\vdash \tau$, so $\vdash \overline{Prov}(\overline{\ulcorner\neg\psi\urcorner}, \overline{0})$ or $\ldots$ or $\vdash \overline{Prov}(\overline{\ulcorner\neg\psi\urcorner}, \overline{n-1})$ hence $Prov(\ulcorner\neg\psi\urcorner, i)$ for some $i < n \Rightarrow \vdash \neg\psi \Rightarrow$ **PA** is inconsistent.
(ii) Suppose $\vdash \neg\psi$ then $\vdash \forall y[\ \overline{Prov}(\overline{\ulcorner\neg\psi\urcorner}, y) \rightarrow \exists z < y\ \overline{Prov}(\overline{\ulcorner\psi\urcorner}, z)\ ]$ also $\vdash \neg\psi \Rightarrow Prov(\ulcorner\neg\psi\urcorner, n)$ for some $n \Rightarrow \vdash \overline{Prov}(\overline{\ulcorner\neg\psi\urcorner}, \overline{n})$ for some $n \Rightarrow \vdash \exists z < \overline{n}\overline{Prov}(\overline{\ulcorner\psi\urcorner}, z) \Rightarrow$ (as in the above) $Prov(\ulcorner\psi\urcorner, k)$ for some $k < n$, so $\vdash \psi \Rightarrow$ **PA** is inconsistent. □

We have seen that truth in $\mathbb{N}$ does not necessarily imply provability in **PA** (or any other (recursively enumerable) axiomatizable extension). However, we have seen that **PA** IS Σ_1^0 complete, so truth and provability still coincide for simple statements.

Definition 7.7.4 *A theory T (in the language of* **PA***) is called Σ_1^0-sound if $T \vdash \varphi \Rightarrow \mathbb{N} \vDash \varphi$ for Σ_1^0-sentences φ.*

We will not go into the intriguing questions of the foundations or philosophy of mathematics and logic. Accepting the fact that the standard model $\mathbb{N}$ is a model of **PA**, we get consistency, soundness and Σ_1^0-soundness for free. It is an old tradition in proof theory to weaken assumptions as much as possible, so it makes sense to see what one can do without any semantic notions. The interested reader is referred to the literature.

We now present an alternative proof of the incompleteness theorem. Here we use the fact that **PA** is Σ_1^0 -sound.

Theorem 7.7.5 PA *is incomplete.*

Consider an RE set X which is not recursive. It is semi-represented by a a Σ_1^0 formula $\varphi(x)$. Let $Y = \{n | \mathbf{PA} \vdash \varphi(\overline{n})\}$.
By Σ_1^0-completeness we get $n \in X \Rightarrow \mathbf{PA} \vdash \varphi(\overline{n})$. Since Σ_1^0 -soundness implies consistence, we also get $\mathbf{PA} \vdash \neg\varphi(\overline{n}) \Rightarrow n \notin X$, hence $Y \subseteq X^c$. The provability predicate tells us that Y is RE. Now X^c is not RE, so there is a number k with $k \in (X \cup Y)^c$. For this number k we know that $\mathbf{PA} \nvdash \neg\varphi(\overline{k})$ and also $\mathbf{PA} \nvdash \varphi(\overline{k})$, as $\mathbf{PA} \vdash \varphi(\overline{k})$ would imply by Σ_1^0 -soundness that $k \in X$. As a result we have established that $\neg\varphi(\overline{k})$ is true but not provable in **PA**, i.e. **PA** is incomplete. □

We almost immediately get the undecidability of **PA**.

Theorem 7.7.6 PA *is undecidable.*

Proof. Consider the same set $X = \{n | PA \vdash \varphi(\overline{n})\}$ as above. If **PA** were decidable, the set XS would be recursive. Hence **PA** is undecidable. □

Note we get the same result for any axiomatizable Σ_1^0 -sound extension of **PA**. For stronger results see Smorynski's *Logical number theory*

that f with $f(n) = \ulcorner \varphi(\overline{n}) \urcorner$ is pr. rec.

Remarks. The Gödel sentence γ "I am not provable" is the negation of a strict Σ_1^0-sentence (a so-called Π_1^0-sentence). Its negation cannot be true (why ?). So **PA** $+ \neg\gamma$ is not Σ_1^0-sound.

We will now present another approach to the undecidability of arithmetic, based on effectively inseparable sets.

Definition 7.7.7 *Let φ and ψ be existential formulas: $\varphi = \exists x \varphi'$ and $\psi = \exists x \psi'$. The witness comparison formulas for φ and ψ are given by:*
$\varphi \leq \psi := \exists x\big(\varphi'(x) \wedge \forall y < x \ \neg\psi'(y)\big)$
$\varphi < \psi := \exists x\big(\varphi'(x) \wedge \forall y \leq x \ \neg\psi'(y)\big)$.

Lemma 7.7.8 (Informal Reduction Lemma) *Let φ and ψ be strict Σ_1^0, $\varphi_1 := \varphi \leq \psi$ and $\psi_1 := \psi < \varphi$. Then*
(i) $\mathbb{N} \models \varphi_1 \to \varphi$
(ii) $\mathbb{N} \models \psi_1 \to \psi$
(iii) $\mathbb{N} \models \varphi \vee \psi \leftrightarrow \varphi_1 \vee \psi_1$
(iv) $\mathbb{N} \models \neg\ (\varphi_1 \wedge \psi_1)$.

Proof. Immediate from the definition. □

Lemma 7.7.9 (Formal Reduction Lemma) *Let φ, ψ, φ_1 and ψ_1 be as above.*
(i) $\vdash \varphi_1 \to \varphi$
(ii) $\vdash \psi_1 \to \psi$
(iii) $\mathbb{N} \models \varphi_1 \Rightarrow \ \vdash \varphi_1$
(iv) $\mathbb{N} \models \psi_1 \Rightarrow \ \vdash \psi_1$
(v) $\mathbb{N} \models \varphi_1 \Rightarrow \ \vdash \neg\psi_1$
(vi) $\mathbb{N} \models \psi_1 \Rightarrow \ \vdash \neg\varphi_1$
(vii) $\vdash \neg(\varphi_1 \wedge \psi_1)$.
Proof. (i)—(iv) are direct consequences of the definition and Σ_1^0-completeness.
(v) and (vi) are exercises in natural deduction (use $\forall uv(u < v \vee v \leq u)$) and (vii) follows from (v) (or (vi)). □

Theorem 7.7.10 (Undecidability of PA) *The relation $\exists y Prov(x, y)$ is not recursive. Popular version: $\vdash$ is not decidable for* **PA**.

Proof. Consider two effectively inseparable recursively enumerable sets A and B with strict Σ_1^0-defined formulas $\varphi(x)$ and $\psi(x)$. Define $\varphi_1(x) := \varphi(x) \leq \psi(x)$ and $\psi_1(x) := \psi(x) < \varphi(x)$.

$$\begin{aligned}
\text{then } n \in A &\Rightarrow \mathbb{N} \vDash \varphi(\overline{n}) \wedge \neg\psi(\overline{n}) \\
&\Rightarrow \mathbb{N} \vDash \varphi_1(\overline{n}) \\
&\Rightarrow \vdash \varphi_1(\overline{n}) \\
\text{and } n \in B &\Rightarrow \mathbb{N} \vDash \psi(\overline{n}) \wedge \neg\varphi(\overline{n}) \\
&\Rightarrow \mathbb{N} \vDash \psi_1(\overline{n}) \\
&\Rightarrow \vdash \neg\varphi_1(\overline{n}).
\end{aligned}$$

Let $\widehat{A} = \{n \mid \vdash \varphi_1(\overline{n})\}$, $\widehat{B} = \{n \mid \vdash \neg\varphi_1(\overline{n})\}$ then $A \subseteq \widehat{A}$ and $B \subseteq \widehat{B}$. **PA** is consistent, so $\widehat{A} \cap \widehat{B} = \emptyset$. $\widehat{A}$ is recursively enumerable, but because of the effective inseparability of A and B not recursive. Suppose that $\{\ulcorner\sigma\urcorner \mid \vdash \sigma\}$ is recursive, i.e. $X = \{k \mid Form(k) \wedge \exists z Prov(k, z)\}$ is recursive. Consider f with $f(n) = \ulcorner\varphi_1(\overline{n})\urcorner$, then $\{n \mid \exists z Prov\left(\ulcorner\varphi_1(\overline{n})\urcorner, z\right)\}$ is also recursive, i.e. $\widehat{A}$ is a recursive separator of A and B. Contradiction. Thus X is not recursive. □

From the indecidability of PA we immediately get once more the incompleteness theorem:

Corollary 7.7.11 PA *is incomplete.*

Proof. (a) If **PA** were complete, then from the general theorem "complete axiomatizable theories are decidable" it would follow that **PA** was decidable. (b) Because $\widehat{A}$ and $\widehat{B}$ are both recursively enumerable, there exists a n with $n \notin \widehat{A} \cup \widehat{B}$, i.e. $\nvdash \varphi(\overline{n})$ and $\nvdash \neg\varphi(\overline{n})$. □

Remark. The above results are by no means optimal; one can represent the recursive functions in considerably weaker systems, and hence prove their incompleteness. There are a number of subsystems of PA which are finitely axiomatizable, for example the Q of Raphael Robinson, (cf. Smorynski, *Logical number theory*, p. 368 ff.), which is incomplete and undecidable. Using this fact one easily gets

Corollary 7.7.12 (Church's theorem) *Predicate logic is undecidable.*

Proof. Let $\{\sigma_1, \ldots, \sigma_n\}$ be the axioms of Q, then $\sigma_1, \ldots, \sigma_n \vdash \varphi \Leftrightarrow$
$\vdash (\sigma_1 \wedge \ldots \wedge \sigma_n) \rightarrow \varphi$. A decision method for predicate logic would thus provide one for Q. □

Remark. (1) Since **HA** is a subsystem of **PA** the Gödel sentence γ is certainly independent of **HA**. Therefore $\gamma \vee \neg\gamma$ is not a theorem of **HA**. For if **HA** $\vdash \gamma \vee \neg\gamma$, then by the disjunction property for **HA** we would have **HA** $\vdash \gamma$ or **HA** $\vdash \neg\gamma$, which is impossible for the Gödel sentence.. Hence we have a specific theorem of **PA** which is not provable in **HA**.
(2) Since **HA** has the existence property, on can go through the first version of the proof of the incompleteness theorem, while avoiding the use of ω-consistency.

Exercises

1. Show that f with $f(n) = \ulcorner t(\overline{n}) \urcorner$ is pr. rec.

2. Show that f with $f(n) = \ulcorner \varphi(\overline{n}) \urcorner$ is pr. rec.

3. Find out what $\varphi \to \varphi \leq \varphi$ means for a φ as above.

Bibliography

The following books are recommended for further reading:

J. Barwise (ed). *Handbook of Mathematical Logic.* North-Holland Publ. Co., Amsterdam 1977.

G. Boolos. *The Logic of Provability.* Cambridge University Press, Cambridge 1993.

E. Börger. *Computability, Complexity, Logic.* North-Holland Publ. Co., Amsterdam 1989.

C.C. Chang, J.J. Keisler. *Model Theory.* North-Holland Publ. Co., Amsterdam 1990

D. van Dalen. Intuitionistic Logic. In: Gabbay, D. and F. Guenthner (eds.) *Handbook of Philosophical Logic. 5.* (Second ed.) Kluwer , Dordrecht 2002, 1–114.

M. Davis. *Computability and Unsolvability.* McGraw Hill, New York 1958.

M. Dummett. *Elements of Intuitionism.* Oxford University Press, Oxford, 1977 (second ed. 2000).

J.Y. Girard. *Proof Theory and Logical Complexity. I.* Bibliopolis, Napoli 1987.

J.Y. Girard, Y. Lafont, P. Taylor. *Proofs and Types.* Cambridge University Press, Cambridge 1989.

P. Hinman *Recursion-Theoretic Hierarchies.* Springer, Berlin 1978.

S.C. Kleene. *Introduction to meta-mathematics.* North-Holland Publ. Co., Amsterdam 1952.

S. Negri, J. von Plato *Structural Proof Theory.* Cambridge University Press, Cambridge 2001.

P. Odifreddi. *Classical Recursion Theory.* North-Holland Publ. Co. Amsterdam 1989.

J.R. Shoenfield. *Mathematical Logic.* Addison and Wesley, Reading, Mass. 1967.

C. Smoryński. *Self-Reference and Modal Logic.* Springer, Berlin. 1985.

J.R. Shoenfield. *Recursion Theory.* Lecture Notes in Logic 1. Springer , Berlin 1993.

C. Smoryński. *Logical Number Theory I.* Springer, Berlin 1991. (Volume 2 is forthcoming).

K.D. Stroyan, W.A.J. Luxemburg. *Introduction to the theory of infinitesimals.* Academic Press, New York 1976.

A.S. Troelstra, D. van Dalen. *Constructivism in Mathematics I, II.* North-Holland Publ. Co., Amsterdam 1988.

A.S. Troelstra, H. Schwichtenberg *Basic Proof theory.* Cambridge University Press, Cambridge 1996.

Index