Utrecht University

Bachelor Thesis
7.5 ECTS

Hilbert's 10th Problem

Author:
Richard Dirven
5686091

Supervisor:
Dr. Jaap van Oosten

A thesis submitted in fulfilment of the requirements for the degree of Bachelor of Science

in the

Department of Mathematics
Faculty of Science

June 17, 2021

"Mathematics knows no races or geographic boundaries; for mathematics, the cultural world is one country."

David Hilbert

Contents

List of Figures

List of Mathematical Notation

$\mathbb{N}^+$	The positive natural numbers
$\mathrm{dom}(f)$	The domain of a function f
$f\vert_A$	Restrict the domain of function f to A
$\subsetneq$	A proper subset
$\setminus$	Set minus
$\min$	The minimalization function
$\max$	The maximization function
$\overline{A}$	The complement of a set A
$\lfloor x \rfloor$	The floor function, with codomain $\mathbb{Z}$
Id	The identity function, sending every element of its set to itself
$\gcd$	The greatest common divisor of two integers
$\div$	Integer division, see [23]
I_n	The $n \times n$ identity matrix
$\Lambda_{i,j}$	The ith row and jth column of a matrix Λ

Dedicated to all people who forgave my mistakes, kept believing in me and pushed me to be better. To all people who believe in cooperation instead of competition.

Part I

Computability Theory

1. Register Machines & Computability

To get an understanding of when a function is computable and when it is not, it is important to rigorously define the notion of *computability*. It is often 'explained' that a function is computable if there exists an algorithm for it. However, for a mathematician, that is not very satisfying, as this just defines computability in terms of something undefined. Intuitively this might all be clear, but if one aims to understand the boundaries of what is and what is not computable, one needs clear definitions. In this chapter a register machine and the notion of a program will be rigorously defined, to enable the definition of a computable function. The theory of this chapter is based on the theory in [19].

1.1 Register Machines

One formalism to describe computability is that of a register machine. Note that there are other ways to define computability, for instance with λ-calculus [5] or with the original Turing machine [2, Ch. 3], but these definitions have been shown to be equivalent.

1.1.1 An overview

Before defining a computation, an understanding of a register machine is necessary. A register machine consists, as the name suggests, out of a fixed number n of registers, which can be considered as boxes containing a natural number each. This can be compared to memory on a typical von Neumann machine [18]. Each register R_i contains a value $r_i \in \mathbb{N}$. Thus, in contrast to a normal computer where memory overflows can occur, these registers can contain arbitrary large values. A visual interpretation of the registers is depicted in figure 1.1. A computer without means to manipulate its memory is nothing more than a storage device. The same holds true for the register machine, which also needs means to manipulate its memory. This is done using a program P, which is a finite ordered list of commands, where each command has one of two shapes:

1. $r_i^+ \Rightarrow n$, which means: add 1 to r_i and move to command n;
2. $r_i^- \Rightarrow n, m$, which means: if $r_i > 0$, subtract 1 from r_i and move to command n. If $r_i = 0$, meaning 1 cannot be subtracted, move to command m.

The numbers n and m will be referred to as instruction pointers, for obvious reasons. A program will halt when an instruction pointer points to $m = |P| + 1$, where $|P|$

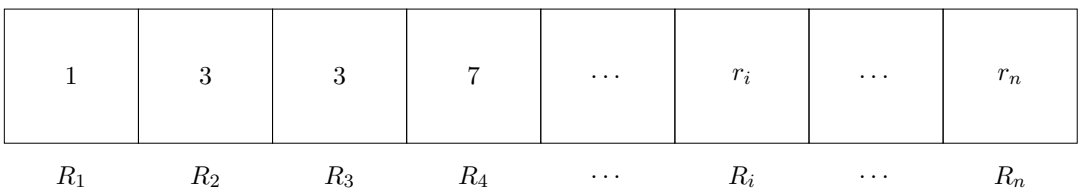


Figure 1.1: Simple graphical representation of a register machine

denotes the number of instructions in program P. This will be also referred to as the STOP-instruction. To be a valid program it must be that all instruction pointers point to values m such that $m \leq |P| + 1$. In contrast to a classical von Neumann machine, where a program needs to be stored in memory (and thus self modifying code can exist), the program of a register machine exists separate from the registers.

Example 1.1 *Some basic operations can now be programmatically described.*

i) Emptying a register:

$E(i) :=$
1 $r_i^- \Rightarrow 1, 2$

ii) Moving data from R_i to R_j:

$M(i, j) :=$
1 $r_i^- \Rightarrow 2, 3$
2 $r_j^+ \Rightarrow 1$

iii) Moving data from R_i to R_j and R_k:

$M(i, j, k) :=$
1 $r_i^- \Rightarrow 2, 4$
2 $r_j^+ \Rightarrow 3$
3 $r_k^+ \Rightarrow 1$

iv) Copying data from R_i to R_j:

$C(i, j) :=$
1 $r_i^- \Rightarrow 2, 4$
2 $r_j^+ \Rightarrow 3$
3 $r_k^+ \Rightarrow 1$
4 $r_k^- \Rightarrow 5, 6$
5 $r_i^+ \Rightarrow 4$

v) Infinitely add one to a register:

1 $r_i^+ \Rightarrow 1$

Note that the program listed in v) never halts, therefore it must not be that a program *terminates*.

1.1.2 Formalization

With the previous section in mind, the soon to be introduced rigor becomes much more understandable.

Definition 1.1 *Let P be a program for the register machine and $a_1, \ldots, a_k \in \mathbb{N}$. A computation of the register machine with program P and input $a_1, \ldots, a_k$ is a list of $l + 1$-tuples, under two conditions.*

1) It must be that $l + 1 \geq k$ and the tuple $(n_1, r_1^1, \ldots, r_l^1) = (1, a_1, \ldots, a_k, 0, \ldots, 0)$;

2) Whenever $n_i = m$, the ith tuple is written as $(m, r_1^i, \ldots, r_l^i)$ and one of four must occur:

i) Program P does not have a mth command, thus this ith tuple is the last tuple and $m = |P| + 1$, the STOP-instruction;

ii) The mth command is $r_j^+ \Rightarrow u$ and $j \leq l$. Then the next tuple is

$$(n_{i+1}, r_1^{i+1}, \ldots, r_l^{i+1}) = (u, r_1^{i+1}, \ldots, r_{j-1}^i, r_j^i + 1, r_{j+1}^i, \ldots, r_l^i);$$

iii) The mth command is $r_j^- \Rightarrow u, v$, $j \leq l$ and $r_j^i > 0$, such that the next tuple is

$$(n_{i+1}, r_1^{i+1}, \ldots, r_l^{i+1}) = (u, r_1^{i+1}, \ldots, r_{j-1}^i, r_j^i - 1, r_{j+1}^i, \ldots, r_l^i);$$

iv) The mth command is $r_j^- \Rightarrow u, v$, $j \leq l$ and $r_j^i = 0$, such that the next tuple is

$$(n_{i+1}, r_1^{i+1}, \ldots, r_l^{i+1}) = (v, r_1^{i+1}, \ldots, r_l^i).$$

This definition is rather long and technical but can be summarized as follows. An element $(n_i, r_1^i, \ldots, r_l^i)$ from the list of $l+1$-tuples in the computation resembles the ith step of the computation. Moreover, the n_ith command of program P will be executed next and the values in the registers $R_1, \ldots, R_n$ are currently $r_1^i, \ldots, r_l^i$. More will be clarified in the example below.

Example 1.2 *The program $M(i, j)$ listed in ii) of the example before will be run on a machine that has two registers R_1 and R_2. Let $r_2 = 2$, so that $M(2, 1)$ is run. Then the computation associated with this is shown below.*

Tuple	*Next instruction*
(1, 0, 2)	$r_2^- \Rightarrow 2, 3$
(2, 0, 1)	$r_1^+ \Rightarrow 1$
(1, 1, 1)	$r_2^- \Rightarrow 2, 3$
(2, 1, 0)	$r_1^+ \Rightarrow 1$
(1, 2, 0)	$r_2^- \Rightarrow 2, 3$
	STOP

As $r_2 = 0$ and there is no command 3 the computation is done.

As stated before, a program does not necessarily terminate and when it does not, it is hard to reason about results or output of the program.

Definition 1.2 *Let $\mathcal{C} = (n_i, r_1^i, \ldots, r_l^i)_{i=1}^K$ be a finite computation; a list consisting out of K elements. The last element of $\mathcal{C}$ is then $(n_K, r_1^K, \ldots, r_l^K)$ and the output of computation $\mathcal{C}$ is defined as r_1^K.*

Example 1.3 *The output of the computation in example 1.2 is 2.*

One might have started to wonder why the number of registers is not really defined. As it turns out, once there are enough of them, adding extra registers does not influence the output of the program. This is formalised in the following remark.

Remark 1.1 *If $(n_i, r_1^i, \ldots, r_l^i)_{i \geq 1}$ is a computation with program P, then so is the computation with $l' \geq l$-tuples $(n_i, r_1^i, \ldots, r_l^i, 0, \ldots, 0)_{i \geq 1}$. Thus, once length is fixed, computations are* unique *implying that register machines are* deterministic*. Consequently, the list with one l-tuple $(1, a_1, \ldots, a_k, 0, \ldots, 0)$ is the unique computation with the empty program $\emptyset$ and input $a_1, \ldots, a_k$.*

To finish this subsection, some notation is introduced. A program P, running on input $\vec{a} = (a_1, \ldots, a_k)$, outputting $\vec{b} = (b_1, \ldots, b_m)$, is denoted as follows

$$\begin{array}{ccc} (a_1, \ldots, a_k) & & \vec{a} \\ \Big\Downarrow P & \text{or} & \Big\Downarrow P \\ (b_1, \ldots, b_m) & & \vec{b} \end{array}.$$

This is called a program diagram.

1.1.3 Closure Properties

It would be desirable to have ways to combine programs to create new programs. One might have noticed from example 1.1 that the first three instructions of $C(i,j)$ are almost the same as in program $M(i,j,k)$ and the two after that are very similar to those in $M(i,j)$. This combination of programs can be defined generally.

Definition 1.3 *Let P and Q be programs where P has k elements. Its composition PQ is defined as follows. First construct Q' from Q by add k to all command numbers. The list by listing all elements of P in order and then Q' is the composition PQ.*

Given this definition it is now possible to write $C(i,j)$ by applying composition on programs $M(i,j,k)$ and $M(k,j)$.

Example 1.4 *To calculate $M(i,j,k)M(k,j)$, modify $M(k,j)$ to $M(k,j)'$ such that it will look like*

$4 \quad r_k^- \Rightarrow 5, 6$
$5 \quad r_i^+ \Rightarrow 4.$

All of the arguments are offset by three because $M(i,j,k)$ has three instructions Indeed, $C(i,j) = M(i,j,k)M(k,j)$, retrieved by listing $M(i,j,k)$ and then $M(k,j)'$.

Having established a notion of composition on programs, some facts about the algebra of programs under composition can be proven. First of all, see that programs, upon fixing register count l, are closed under composition. But that is not the only pleasant algebraic property of program composition.

Lemma 1.2 *Program composition is associative.*

Proof. Let n_1, n_2 and n_3 be arbitrary instruction pointers from programs P, Q and R respectively. As composition only modifies the instruction pointers, it suffices to look at where they get mapped to when composing P with QR or PQ with R. Composition PQ modifies n_2 to $|P| + n_2$ and $(PQ)R$ then modifies n_3 to $(|P| + |Q|) + n_3$. The program composition QR modifies n_3 to $|Q| + n_3$ and pre-composing P with QR further modifies n_3 to $|P| + (|Q| + n_3)$ and it modifies n_2 to $|P| + n_2$. Observe that n_1 stays the same, n_2 always gets send to $|P| + n_2$ and by associativity of addition n_3 always gets send to $|P| + |Q| + n_3$. □

It would be desirable if program composition also has a neutral element. Luckily, neutral elements are often easy to find.

Lemma 1.3 *Let P be a program and $\emptyset$ be the empty program. Then $\emptyset P = P = P\emptyset$.*

Proof. As $\emptyset$ is an empty, zero-length program, all instruction pointers and counters of P remain unchanged, thus $P' = P$. Listing all the elements of $\emptyset$ first does nothing, thus $\emptyset P = P$. When calculating $P\emptyset$, all instruction pointers of $\emptyset$ get changed. However, there are no instruction pointers in the empty program. Hence $P\emptyset = P$, proving the desired result. □

Program composition has a neutral element and is associative, meaning they form a monoid.

1.2 Computable Functions

1.2.1 Introduction

Having established a concise definition of what a computation is, the next goal is to establish a notion of computable functions. Let $f\colon A \to \mathbb{N}$, where $A \subset \mathbb{N}^k$ and k is some positive integer. Then f is a k-ary partial function. When $\mathrm{dom}(f) = \mathbb{N}^k$, f is a total function. For $k = 1, 2, 3$ one can speak of unary, binary and ternary functions respectively.

Definition 1.4 *Let f be a k-ary function, then f is said to be (RM-)computable if there exists a program P such that for all $\vec{a} \in \mathbb{N}^k$ one of two applies:*

1) *$\vec{a} \in \mathrm{dom}(f)$: program P on input $\vec{a}$ terminates and the output of this computation is $f(\vec{a})$;*

2) *$\vec{a} \notin \mathrm{dom}(f)$: there is no finite computation with P on input $\vec{a}$.*

The definition will be seen in practice in the following example.

Example 1.5 *Define $f\colon \mathbb{N} \to \mathbb{N}$ as follows: $f(a) = 0$, thus f is the* zero function. *Let P be the program listed in i), for register R_1. Clearly P computes f, as P always outputs zero, independent of its input and therefore f is RM-computable.*

The second part of the definition plays an interesting role, shown by the example below.

Example 1.6 *Let f be an arbitrary RM-computable function and P the program to compute it. Moreover, let $A \subsetneq \mathrm{dom}(f)$. The restriction $f|_A$ need not be RM-computable, because for any $a \in \mathrm{dom}(f) \setminus A$ the program P still terminates (and has $f(a)$ as its output).*

By composing programs one can create new programs. One of such programs is the following.

Lemma 1.4 *Let f be a k-ary function which is computable with program P. Then there exists a program $\widetilde{P}$ such that for all $\vec{a} \in \mathrm{dom}(f)$, we have that*

$$\begin{array}{c}(a_1, \ldots, a_k) \\ \Big\Downarrow \widetilde{P} \\ (f(\vec{a}), a_1, \ldots, a_k)\end{array} .$$

For a proof see appendix A.1.

1.2.2 Closure Properties

Not only programs are closed under composition, but also computable functions.

Definition 1.5 *Suppose $g_1, g_2, \ldots, g_l\colon \mathbb{N}^k \to \mathbb{N}$ and $h\colon \mathbb{N}^l \to \mathbb{N}$ are computable functions. A function $f\colon \mathbb{N}^k \to \mathbb{N}$ is said to be defined from composition of $g_1, \ldots, g_l$ and h if*

1) *the domain of f is*

$$\{\vec{a} \in \mathbb{N}^k \mid \vec{a} \in \bigcap_{i=1}^{k} \mathrm{dom}(g_i) \text{ and } (g_1(\vec{a}), \ldots, g_l(\vec{a})) \in \mathrm{dom}(h)\};$$

2) if $\vec{a} \in \mathrm{dom}(f)$ it holds that $f(\vec{a}) = h(g_1(\vec{a}), \ldots, g_l(\vec{a}))$.

Lemma 1.5 *The set of computable functions is closed under composition.*

For a proof, see appendix A.2. There are even more closure properties of computable functions, the most important of which is using recursion. In chapter 2 this will be further explored.

Definition 1.6 *Let g, h and f be k-ary, $k+2$-ary and $k+1$-ary partial functions respectively. Define*

$$\begin{aligned} F_0 &= \{(0, x_0, \ldots, x_k) \mid (x_1, \ldots, x_k) \in \mathrm{dom}(g)\} \\ F_{y+1} &= \{(y+1, x_0, \ldots, x_k) \mid (y, x_1, \ldots, x_k) \in F_y \\ &\qquad \text{and } (y, f(y, x_1, \ldots, x_k), x_1, \ldots, x_k) \in \mathrm{dom}(h)\} \end{aligned}$$

The $k+1$-ary function f is said to be defined from the primitive recursion of g and h when

1) The domain of f is $\cup_{i=0}^{\infty} F_i$;

2) $f(0, x_1, \ldots, x_k) = g(x_1, \ldots, x_k)$;

3) $f(y+1, x_1, \ldots, x_k) = h(y, f(y, x_1, \ldots, x_k), x_1, \ldots, x_k)$.

This definition is not very intuitive at glance. This can be seen as a `for`-loop, where one starts at $f(0, \vec{a})$ and uses the result of computation $f(y, \vec{a})$ to compute the result of $f(y+1, \vec{a})$. In a for loop one is also able to use `i`, the looping parameter, which translates to h being able to use the parameter y provided.

Example 1.7 *It is possible to define exponentiation by primitive recursion. Then first define*

$$\begin{aligned} g\colon \mathbb{N} &\to \mathbb{N} & \qquad h\colon \mathbb{N}^3 &\to \mathbb{N} \\ x &\mapsto 1 & \qquad (x, y, z) &\mapsto y \cdot z. \end{aligned}$$

Now let f be defined by the primitive recursion of g and h. The, for example,

$$f(2, x) = h(1, f(1, x), x) = x \cdot f(1, x) = x \cdot h(0, f(0, x), x) = x \cdot (x \cdot g(x)) = x \cdot (x \cdot 1) = x^2.$$

It can be shown with induction that f is the exponentiation function x^y.

One might wonder whether f is computable given that it is defined from primitive recursion of computable functions g and h. It turns out that is true, which is proven in the following lemma.

Lemma 1.6 *The set of computable functions is closed under primitive recursion.*

A proof is again listed in the appendix A.3. There is one last operation under which computable functions are closed. This operation is lesser known outside of the "computability scene".

Definition 1.7 *Let f and g be k-ary respectively $k+1$-ary functions. Define f such that $\vec{x} \in \mathrm{dom}(f)$ if there is a y such that $g(y, \vec{x}) = 0$. Moreover for all $0 \leq j \leq y$ the $k+1$-tuples $(j, \vec{x})$ are contained in $\mathrm{dom}(g)$. In that case*

$$f(\vec{x}) = \min\{y \mid g(y, \vec{x}) = 0\}.$$

The function f is said to be defined from minimalization from g.

Lemma 1.7 *The set of computable functions is closed under minimalization.*

A proof is shown in appendix A.4. Take note that the second part of definition 1.4 is important for minimalization, because a RM computing $f(\vec{a})$, is allowed to search forever for a minimal value, iterating over all the naturals. The program will never halt if there is not an i for which $g(i, \vec{a}) = 0$. But that just emphasizes that $\vec{a} \notin \mathrm{dom}(f)$.

2. Primitive Recursive Functions

2.1 Introduction

So far, the reader has been introduced with *computations* and using those *computable functions* have been defined; a layer which builds on the definition of programs. Another layer will now be added, building upon the definition of computable functions. This is done analogously to [19], as this chapter is based on those lecture notes. To do so, some extra notation needs to be introduced to avoid ambiguity. One might have been in the situation, during calculus class for instance, that there are so many symbols in an equation, that it is not immediately clear which symbols stand for parameters and which stand for variables. For instance, look at the linear[1] function

$$ax + by.$$

This could mean multiple things:

1) A function of (x, y), $\mathbb{N}^2 \to \mathbb{N}$, where a and b are parameters;
2) However, just as valid, it could be a function of (a, b), $\mathbb{N}^2 \to \mathbb{N}$, where x and y are the parameters;
3) But it could also be a function of (y, x);
4) Or even more unconvential, a function of (a, x, y): $\mathbb{N}^3 \to \mathbb{N}$, without parameters;

In fact all permutations of 0, 1, 2, 3 and 4 variables constitute to a reasonable definition of a function. One might start to wonder how one even has been able to do mathematics while so much ambiguity exists! In computability theory this is extra annoying, as functions and defining functions are what this theory is all about. Therefore, the lambda notation is being used. It is best shown in practice, thus the four examples above will be shown.

1) $\lambda xy.ax + by$;
2) $\lambda ab.ax + by$;
3) $\lambda yx.ax + by$;
4) $\lambda abxy.ax + by$.

To insert values for the variables, say $x = 3$ and $y = 5$, write $(\lambda xy.ax + by)(3, 5)$. The difference between the first and third notation becomes evident here.

Definition 2.1 *The class of primitive recursive functions consists out of functions $\mathbb{N}^k \to \mathbb{N}$, where k can be any non-negative integer. This class is generated by:*

1) *the 1-ary zero function* $\text{Zero} = \lambda x.0$;
2) *the 1-ary successor function* $S = \lambda x.x + 1$;
3) *the k-ary projection functions* $\Pi_i^k = \lambda x_1 \dots x_k.x_i$;
4) *all compositions of two primitive recursive functions;*

[1] Linear when we consider, as is often convention, a and b to be constants and x and y to be variables.

5) all primitive recursions on two primitive recursive functions.

The first three functions are referred to as basic primitive recursive functions

Earlier on primitive recursion of two functions has been explained and used in an example, so the construction of new primitive recursive functions should be clear. Do note that this definition does not contain minimalization. A small example is shown to further explain the definition of primitive recursive functions.

Example 2.1 *The constant function $f_k = \lambda x.k$, where $k \in \mathbb{N}$ is primitive recursive. This can be shown by induction. First note that f_k is the zero-function which is primitive recursive by definition. For the induction step, assume that $f_k(x)$ is primitive recursive. See that $f_{k+1}(x) = k + 1 = S(k) = S(f_k(x))$. Because S is primitive recursive and f_k is by our assumption, f_{k+1} is too by applying composition.*

Primitive recursive functions have two very attractive properties.

Lemma 2.1 *All primitive recursive functions are total.*

Proof. First see that items 1) to 3) are trivially total. Now assume that f is defined from composition of total functions $g_1, \ldots, g_l \colon \mathbb{N}^k \to \mathbb{N}$ and $h \colon \mathbb{N}^l \to \mathbb{N}$. Each g_i is total and therefore $\mathrm{dom}(g_i) = \mathbb{N}^k$. Additionally h is total, hence for arbitrary $\vec{a}$ $(g_1(\vec{a}), \ldots, g_l(\vec{a})) \in \mathrm{dom}(h)$. Ergo, f, by the definition of its domain, is total.
If f is defined from the primitive recursion of total functions $g \colon \mathbb{N}^k \to \mathbb{N}$ and $h \colon \mathbb{N}^{k+2} \to \mathbb{N}$, let $\vec{a} \in \mathbb{N}^k$ be arbitrary, so that $f(0, \vec{a}) = g(\vec{a})$. As g is total, this is defined, thus $(0, \vec{a}) \in \mathrm{dom}(f)$. Let the induction hypothesis be that $(y, \vec{a}) \in \mathrm{dom}(f)$. Then $f(y + 1, \vec{a}) = h(y, f(y, \vec{a}), \vec{a})$. By the IH $f(y, \vec{a})$ exists and by totality of h, $(y, f(y, \vec{a}), \vec{a}) \in \mathrm{dom}(h)$. Therefore $(y + 1, \vec{a}) \in \mathrm{dom}\, f$. This holds for arbitrary $\vec{a}$, proving the lemma. □

If a function f is obtained from minimalization of g, then f is not necessarily total. This explains why minimalization is not included in the definition of primitive recursive functions. By including minimalization the class of partial recursive functions is obtained, which is what section 2.3 is all about.

Lemma 2.2 *All primitive recursive functions are computable.*

Proof. For each item listed in definition 2.1 a program or proof is given.

1) The function $E(1)$, from item i) computes the zero-function;

2) The program

   $1 \quad r_1^+ \Rightarrow 2$

   computes S;

3) The following program computes the projection Π_k^i:

$$\begin{array}{c} x_1 \ \ldots \ x_k \\ \Downarrow \\ 0\, x_1 \ \ldots \ x_k \\ \Big\Downarrow C(k,1) \\ x_i\, x_1 \ \ldots \ x_k \end{array}$$

4) This was proven in lemma 1.5;

5) This was proven in lemma 1.6. □

2.2 Important primitive recursive functions

Basic primitive recursive functions act as building blocks, from which lots of new functions can be build. Primitive recursive functions can also be used to induce relations on sets.

Definition 2.2 *Let A be a subset of $\mathbb{N}^k$ for some k, then A is a k-ary relation. Define the function*

$$\chi_A \colon \mathbb{N}^k \to \mathbb{N}$$
$$\vec{x} \mapsto \begin{cases} 1 & \text{if } \vec{x} \in A \\ 0 & \text{else} \end{cases}.$$

Then χ_A is called the characteristic function. A relation is called primitive recursive if the characteristic function is.

Observe that the characteristic function is necessarily total.

2.2.1 Primitive recursiveness of the equality relation

First it is shown that equality is a primitive recursive relation. Most of the proofs are based on versions on [3]. This subsection can be skipped, but take note of the functions being defined in the lemmas. The set associated with the equality relation is

$$E = \{(a, a) \mid a \in \mathbb{N}\}.$$

The primitive recursiveness of E is shown incrementally, demonstrating the power of basic primitive recursive functions as building blocks. All functions are, *naturally*, $\mathbb{N} \to \mathbb{N}$, unless stated otherwise.

Lemma 2.3 *Let $\mathrm{Add}(x, y) = \lambda xy.x + y$, then $\mathrm{Add}(x, y)$ is primitive recursive.*

Proof. See that $\mathrm{Add}(0, y) = 0 + y = y = \Pi^1_1(y)$, which is a basic primitive recursive function. Furthermore $\mathrm{Add}(x + 1, y) = x + 1 + y = \mathrm{Add}(x, y) + 1 = S(\Pi^3_2(x, \mathrm{Add}(x, y), y))$, which is primitive recursive because it is a composition of two basic primitive recursive functions. Conclude that Add is a primitive recursion of two primitive recursive functions and therefore it also primitive recursive. □

Lemma 2.4 *Let $\mathrm{Prd}(x)$ be the predecessor function, that is:*

$$\mathrm{Prd}(x) = \begin{cases} 0 & \text{if } x = 0 \\ x - 1 & \text{if } x > 0 \end{cases}.$$

Then Prd is primitive recursive.

Proof. The function Prd can be defined by primitive recursion of the 0-ary function $\Pi^1_1(0)$ and the projection function $\Pi^2_1(x, y)$. Consequently the function looks like:

$$\mathrm{Prd}(x) = \begin{cases} 0 & \text{if } x = \Pi^1_1(0) \\ \Pi^2_1(x - 1, \mathrm{Prd}(x - 1)) & \text{else} \end{cases}.$$ □

Lemma 2.5 *Define the cut-off subtraction function $\dot{-}$ as*

$$x \dot{-} y = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{if } x < y \end{cases}.$$

Then the cut-off subtraction function is primitive recursive.

Proof. It holds that $x \dot{-} 0 = x = \Pi^1_1(x)$ is prmitive recursive and that $x \dot{-} (y+1) = (x \dot{-} y) - 1 = \mathrm{Prd}(x \dot{-} y) = \Pi^3_2(y, \mathrm{Prd}(x \dot{-} y), x)$ is primitive recursive. Thus $x \dot{-} y$ can be defined by primitive recursion of Π^1_1 and $\lambda xyz.\,\mathrm{Prd}(y)$. □

Lemma 2.6 *The absolute difference function $|x - y|$, defined as*

$$|x - y| = \begin{cases} x - y & \text{if } x \geq y \\ y - x & \text{else} \end{cases},$$

is primitive recursive.

Proof. Realize that $|x - y| = (x \dot{-} y) + (y \dot{-} x)$. Therefore we can write that $|x - y| = \mathrm{Add}(x \dot{-} y, y \dot{-} x)$, which is primitive recursive by composition of primitive recursive functions. □

Theorem 2.7 *The characteristic function of equality, noted as χ_{eq}, is primitive recursive.*

Proof. Note that $x = y$ iff $|x - y| = 0$. Moreover, it holds that $x \dot{-} (x \dot{-} 1)$ is zero when x is zero and one otherwise. Therefore

$$\chi_{\mathrm{eq}}(x, y) = 1 \dot{-} |x - y|.$$ □

Two examples are worked out for extra clarity of this somewhat complex looking function. In the next section some extra notation is introduced, making it somewhat easier to express χ_{eq} intuitively.

Example 2.2 *Let $x = 2$ and $y = 5$, the computation is as follows.*

$$\begin{aligned} \chi_{\mathrm{eq}}(2, 3) &= 1 \dot{-} |2 - 5| \\ &= 1 \dot{-} 3 \\ &= 0 \end{aligned}$$

Example 2.3 *Let both $x, y = 5$, the computation is as follows.*

$$\begin{aligned} \chi_{\mathrm{eq}}(5, 5) &= 1 \dot{-} |5 - 5| \\ &= 1 \dot{-} 0 \\ &= 1 \end{aligned}$$

As a bonus, it is shown in figure 2.1 how all these functions depend on each other.

2.2.2 Powerful building blocks

In the previous subsection a lot of primitive recursive functions were necessary, just to show the primitive recursiveness of the equality relation. In contrast, this section is dedicated to providing general patterns, to construct whole classes of primitive recursive functions all at once. First a connection between primitive recursive relations and their complements is stated.

Lemma 2.8 *Let A denote a primitive recursive relation, then so is $\overline{A}$.*

Proof. Because A is a primitive recursive relation, it has an associated characteristic function χ_A. Define $\chi_{\overline{A}}$ as $\lambda x.1 \dot{-} \chi_A(x)$. If $x \notin A$, then $\chi_A(x) = 0$, thus $\chi_{\overline{A}}(x) = 1$. In the same manner, if $x \in A$, $\chi_A(x) = 1$ giving that $\chi_{\overline{A}}(x) = 0$. □

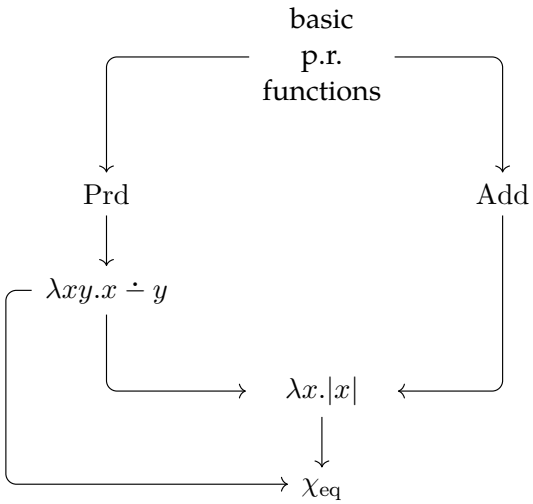


Figure 2.1: Dependency graph of different primitive recursive functions

This immediately begs for some notational convenience.

Definition 2.3 *Let $f : \mathbb{N} \to \{0,1\} \subset \mathbb{N}$. Then its complement is defined as*

$$\overline{f(x)} = 1 \dot{-} f(x).$$

This definition enables one to write $\overline{\chi_A}$ instead of $\chi_{\overline{A}}$ for any characteristic function. From now on, the proofs regarding the primitive recursiveness of a function will be more brief: any function will be written in terms of other primitive recursive functions. It should be straightforward to see whenever primitive recursion or composition is applied.
Another important function is the sign function sgn, tailored for the natural numbers. It is defined as

$$\text{sgn}(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{else} \end{cases}.$$

See that $\text{sgn} = \chi_{\mathbb{N}^+}$ and as its co-domain is $\{0,1\}$, it has a complement $\overline{\text{sgn}}$.

Lemma 2.9 *The function $\text{sgn}(x)$ is primitive recursive.*

Proof. The sign function can be written as $\lambda x.x \dot{-} (x \dot{-} 1)$. □

In the proof of theorem 2.7 this was already being used. Using definition 2.3 χ_{eq} can be written as:

$$\chi_{\text{eq}}(x,y) = \overline{\text{sgn}(|x-y|)}.$$

A lot of functions defined in section 2.2.1 have been defined using cases. This can be generalized with the case function, which is defined as

$$C(x,y,z) = \begin{cases} x & \text{if } z = 0 \\ y & \text{else} \end{cases}.$$

As one might guess, this function is primitive recursive. To show this, however, a closer look at multiplication is needed.

Lemma 2.10 *The function $\lambda xy.x \cdot y$ is primitive recursive.*

Proof. See that $x \cdot 0 = 0 = \text{Zero}(x)$. In addition to that, $x \cdot (y+1) = x \cdot y + y = \Pi_2^3(x, \text{Add}(x \cdot y, y), y)$. □

Lemma 2.11 *The function C is primitive recursive.*

Proof. Write $C(x,y,z)$ as $\lambda xyz.\chi_{\text{eq}}(z,0) \cdot x + \overline{\chi_{\text{eq}}(z,0)} \cdot y$. □

Mathematicians love operations on sequences, it is therefore desirable to show that some sequence manipulations are primitive recursive. Three such series manipulations are shown to be primitive recursive. The most common one, summation:

Lemma 2.12 *Let f be a $k+1$-ary primitive recursive function. Then*

$$\lambda \vec{x} z. \sum_{y<z} f(y, \vec{x})$$

is primitive recursive.

Proof. Note that $(\lambda \vec{x} z. \sum_{y<z} f(y,x))(\vec{x}, 0) = 0 = \text{Zero}(\Pi_1^1(\vec{x}))$ for arbitrary $\vec{x} \in \mathbb{N}^k$, per the empty sum property. Also it holds that

$$(\lambda \vec{x} z. \sum_{y<z} f(y, \vec{x}))(\vec{x}, z+1) = (\lambda \vec{x} z. \sum_{y<z} f(y, \vec{x}))(\vec{x}, z) + f(\vec{x}, z).$$ □

A similar proof for multiplicative series is shown.

Lemma 2.13 *Let f be a $k+1$-ary primitive recursive function. Then*

$$\lambda \vec{x} z. \prod_{y<z} f(y, \vec{x})$$

is primitive recursive.

Proof. First study the function when $z = 0$ and see that $(\lambda \vec{x} z. \prod_{y<z} f(y, \vec{x}))(\vec{x}, 0) = 1$ as a property of the empty product. Moreover,

$$(\lambda \vec{x} z. \prod_{y<z} f(y, \vec{x}))(\vec{x}, z+1) = f(\vec{x}, z)(\lambda \vec{x} z. \prod_{y<z} f(y, \vec{x}))(\vec{x}, z).$$ □

The last series manipulation needs to be properly defined before any properties of it can be demonstrated.

Definition 2.4 *Let χ be a $k+1$-ary characteristic function and fix $c \in \mathbb{N}$, $\vec{x} \in \mathbb{N}^k$. Then the function denoted with and defined as*

$$\mu y < c.\chi(\vec{x}, y) = \begin{cases} c & \text{if } \{y \mid \chi(\vec{x}, y) = 1\} = \emptyset \\ \min\{y \mid \chi(\vec{x}, y) = 1\} & \text{else} \end{cases}$$

is called bounded minimalization.

This minimization operator is used like the λ-function regarding notation and is also referred to as the bounded μ operator. The bounded μ operator is not as 'famous' and used outside the realms of computability theory as the summation and product operators.

Lemma 2.14 *Let χ be a $k+1$-ary primitive recursive characteristic function. Then*

$$\lambda\vec{x}z.(\mu y < z.\chi(\vec{x}, y))$$

is primitive recursive.

Proof. Analogous to the two preceding lemmas: $\lambda\vec{x}z.(\mu y < z.\chi(\vec{x}, y))(\vec{x}, 0) = 0$, for all $\vec{x} \in \mathbb{N}^k$. Furthermore it holds that

$$\lambda\vec{x}z.(\mu y < z.\chi(\vec{x}, y))(\vec{x}, z+1) = \lambda\vec{x}z.(\mu y < z.\chi(\vec{x}, y))(\vec{x}, z) + \prod_{y<z+1} \overline{\chi(\vec{x}, y)}. \qquad \square$$

The second term adds one when there has not been any $y < z+1$ such that $\chi = 1$ and zero otherwise.

This covers the theory about primitive recursive functions. There is a lot more theory regarding those. For proofs about which functions are primitive recursive, see [3]. A good book which includes this topic is [2], which is also a good general introduction to computability theory.

2.3 Partial Recursive Functions

Up to this point, the reader has only been introduced to total functions. Despite them being well behaved, they do not cover enough ground to make a theory about computability. Illustrated by the proof of lemma 1.7, computations need not halt during minimalization. In contrast, computations of total k-ary functions f will always halt, because any k-tuple is contained in $\mathrm{dom}(f)$. Hence the following definitions.

Definition 2.5 *The class of partial recursive functions is generated by:*

1) *all primitive recursive functions;*
2) *minimalization of a partial recursive function (recall definition 1.7);*
3) *the composition $\lambda\vec{x}.f(g(\vec{x}))$, where f is a unary* primitive *recursive function and g is a k-ary partial recursive function.*

Partial recursive functions may also be referred to as μ-recursive functions. The definition begs the question whether there are partial recursive relations. The answer is yes and no.

Definition 2.6 *A relation $A \subset \mathbb{N}^k$ is called recursive if its characteristic function χ_A is partial recursive.*

A partial recursive function is called total recursive or just recursive if it is total. To answer the question; in some sense there are partial recursive relations, but note that χ_A is total by definition. If χ_A is not total then A is a recursively enumerable set , which will be explored further in section 4.2. Consequently all partial recursive relations are total recursive relations and as such, it is just as easy to call them recursive relations. Different kind of functions have now been mentioned, so to clarify, in figure 2.2 their relation is shown. A simple example of a partial recursive function is given.

Example 2.4 *Let χ_A be the characteristic function for the relation $A = \{(x, y) \mid x + y = 1\}$. Moreover define $f(x) = \mu y.[x + y = 1]$. Then f is partial recursive, because it is a composition of minimalization and the primitive recursive function of equality. The domain of f is $\{0, 1\}$.*

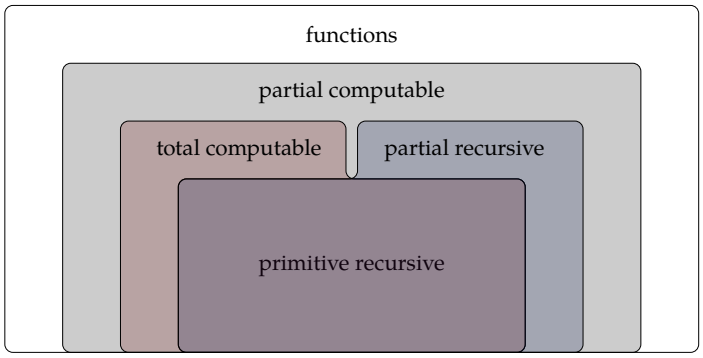


Figure 2.2: Relation between different types of (recursive) functions

To conclude this chapter, a special kind of equality is introduced. This is done, because from this section on some functions can be very similar but defined on another domain.

Definition 2.7 *Let f and g be functions, the Kleene equality $f(x) \simeq g(x)$ means that whenever one of $f(x), g(x)$ is defined so is the other and equality between them holds.*

Again, an example can be very enlightening.

Example 2.5 *Consider $f(x) = x$, $g(x) = \frac{x^2}{x}$ and $h(x) = \frac{x^3}{x^2}$, then $f(x) \not\simeq g(x)$, because $g(0)$ is undefined, but $f(0)$ is not. However $g(x) \simeq h(x)$ because both are undefined in $x = 0$ and agree whenever they are defined.*

3. Tuple Coding

An important tool in computability theory is the encoding of tuples and programs as a natural number. The theory in this chapter is based on the theory from [19]. For instance, it is impossible to output an $\vec{a} \in \mathbb{N}^k$, where $k > 1$, but with encoding it is! In [10, pp. 729–731] the bijection between $\mathbb{N}^2$ and $\mathbb{N}$ is proven.

Definition 3.1 *Let $f : \mathbb{N}^2 \to \mathbb{N}$ such that f is computable and bijective. Then f is said to be a pairing function.*

One such pairing function will be introduced and explored in the next section.

3.1 The Cantor pairing function

Although multiple such pairing functions exist, in this thesis all coding is done with one such function.

Definition 3.2 *Define*

$$\begin{aligned} \pi &: \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\ (x, y) &\mapsto \frac{(x+y)(x+y+1)}{2} + y. \end{aligned}$$

Then this function is called the Cantor pairing function.[1]

The Cantor pairing function has the useful property that both$x, y \leq \pi(x, y)$. In fact, the Cantor pairing function iterates tuples in a neatly fashion, as shown in figure 3.1. The most important property of it will now be proven.

Lemma 3.1 *The Cantor pairing function is bijective.*

Its proof, accompanied with an example is listed in appendix B.1. This proof however, calculates x and y for a given z using the floor function of a square root. Therefore, to reverse the bijection, another definition is more suitable.

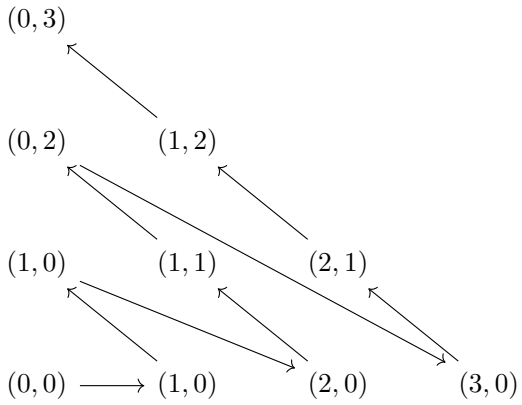


Figure 3.1: Visualization of the Cantor pairing function

[1] Sometimes it is just referred to as *the* pairing function, but others do exist, see [20] and [21] for some examples.

Definition 3.3 *Let z be an arbitrary natural number and let*

$$\pi_1(z) = \mu x \leq z.[\exists y \leq z.\pi(x,y) = z]$$
$$\pi_2(z) = \mu y \leq z.[\exists x \leq z.\pi(x,y) = z]$$

then $\pi_1(z) = x$ and $\pi_2(z) = y$ are the Cantor projections.

One might wonder whether this definition works. Hence the following lemma is stated.

Lemma 3.2 *For arbitrary $z \in \mathbb{N}$ it holds that $\pi(\pi_1(z), \pi_2(z)) = z$.*

Proof. Let $x = \pi_1(z)$ and $y = \pi_2(z)$. Then there exists a y' such that $\pi(x, y') = z$, likewise there exists a x' such that $\pi(x', y) = z$. But π is bijective so $x = x'$ and $y = y'$. Consequently, $\pi(x, y) = z$. □

Now it becomes possible to show that the Cantor pairing function is truly a pairing function.

Lemma 3.3 *The Cantor pairing function and its projections are computable.*

Its proof is listed in appendix B.2. The Cantor pairing function is somewhat surprisingly the only 2^{nd} degree polynomial pairing function, up to permutation of the variables. The proof can be found in [11]. Interestingly, in the same paper the following conjecture is made.

Conjecture 1 *The Cantor pairing function is the only pairing function in polynomial form.*

Later in this thesis more open conjectures will be shown, which provide the reader with ideas for further research.

3.2 Tuple pairing function

Using this pairing function, it is also possible to biject triples, quadruples or k-tuples instead of tuples to the naturals. This is done as follows:

Definition 3.4 *Let $k \in \mathbb{N}^+$, then the kth pairing function $\pi^k : \mathbb{N}^k \to \mathbb{N}$ is defined recursively by letting*

$$\pi^1 = \mathrm{Id}$$
$$\pi^{k+1}(x_1, \ldots, x_k, x_{k+1}) = \pi(\pi^k(x_1, \ldots, x_k), x_{k+1}).$$

The kth projection function $\pi_i^k : \mathbb{N} \to \mathbb{N}$, satisfying

$$\pi^k(\pi_1^k(y), \ldots, \pi_k^k(y)) = y,$$

for arbitrary $y \in \mathbb{N}$, is defined recursively by

$$\pi_1^1 = \mathrm{Id}$$
$$\pi_i^{k+1}(y) = \begin{cases} \pi_i^m(\pi_1(y)) & \text{if } i < k+1 \\ \pi_2(y) & \text{if } i = k+1 \end{cases}.$$

Note that the second pairing function is just referred to as *the pairing function* and similarly that the second projection function is *the projection function*.

Lemma 3.4 *Each π^k is bijective.*

Proof. Clearly for $k = 1, 2$ this is true. Pick any k and assume that the lemma holds for k. See that $\pi^{k+1}(x_1, \ldots, x_{k+1}) = \pi(\pi^k(x_1, \ldots, x_k), x_{k+1})$. By the IH π^k is a bijection between $\mathbb{N}^k$ and $\mathbb{N}$, define $x = \pi^k(x_1, \ldots, x_k)$, then $\pi(x, x_{k+1})$ is a bijection. Ergo π^{k+1} is a bijection. □

Via an analogous way it is straightforward to prove that each π^k is computable. For each k a bijection between $\mathbb{N}^k$ and $\mathbb{N}$ has been established. Recall that a countable union of countable sets is itself is countable, therefore $\cup_{k=0}^{\infty}\mathbb{N}^k$ is also countable. It begs the question whether is possible to build a computable bijection from $\cup_{k=0}^{\infty}\mathbb{N}^k \to \mathbb{N}$. There is such a bijection and this is desirable, because it can be used in a range of proofs.

Definition 3.5 *Let $(x_1, \ldots, x_k) \in \mathbb{N}^k$ and write $(\_)$ for the empty tuple in $\mathbb{N}^0$. The code of a sequence is defined and written as*

$$\begin{aligned} \langle \_ \rangle &= 0 \\ \langle x_0, \ldots, x_{k-1} \rangle &= \pi(k-1, \pi^k(x_0, \ldots, x_{k-1})) + 1 \quad \text{if } k > 0. \end{aligned}$$

By similar reasoning as before, the code of a sequence is bijective and computable. Note however, that the indices start from zero. This is done for convenience and analogously to the convention that the naturals start at 0. Being able to assign a unique natural number to any element of $\cup_{k=0}^{\infty}\mathbb{N}^k$ is quite astonishing. An example is shown below.

Example 3.1 *By the bijective property every 0-vector gets assigned a unique value. Therefore, it holds that*

1. *By definition $\langle \_ \rangle = 0$;*
2. *Continuing on, see that $\langle 0 \rangle = \pi(0, \pi^1(0)) + 1 = 1$. Here it becomes clear why the +1 was added;*
3. *Next, it holds that $\langle 0, 0 \rangle = \pi(1, \pi^2(0, 0)) + 1 = 2$. A pattern seems to emerge;*
4. *Lastly, $\langle 0, 0, 0 \rangle = \pi(2, \pi^3(0, 0, 0)) + 1 = 4$, meaning that not all zeroes are enumerated after each other.*

3.3 Functions on sequences

In this section some fundamental manipulations on sequences are introduced. They demonstrate flexibility of encoding. For instance the length of an encoded sequence can be recovered.

Definition 3.6 *The length of a sequence, encoded as x, can be retrieved by the function $\ell(x)$ and is defined as follows:*

$$\ell(x) = \begin{cases} 0 & \text{if } x = 0 \\ \pi_1(x-1) + 1 & \text{if } x > 0 \end{cases}.$$

Lemma 3.5 *Given an encoded sequence $x = \langle x_0, \ldots, x_{k-1} \rangle$, it holds that $\ell(\langle x \rangle) = k$.*

Proof. Simply apply the definitions of the sequence encoding and the ℓ-function, to get that

$$\begin{aligned}\ell(x) &= \ell(\langle x_0, \ldots, x_{k-1}\rangle)\\ &= \ell(\pi(k-1, \pi^k(x_0, \ldots, x_k) + 1))\\ &= \pi_1(\pi(k-1, \pi^k(x_0, \ldots, x_k))) + 1\\ &= k.\end{aligned}$$

□

With this function it becomes possible to recover individual elements of the encoded sequence.

Definition 3.7 *The element recovering function* $(x)_i$*, recovering the* i*th element of a sequence encoded as* x*, is defined as*

$$(x)_i = \begin{cases}\pi_{i+1}^{\ell(x)}(\pi_2(x-1)) & \text{if } 0 \leq i < \ell(x)\\ 0 & \text{else}\end{cases}.$$

It should be fairly straightforward to see why the element recovering function works.

Lemma 3.6 *Given an encoded sequence* $x = \langle x_0, \ldots, x_i, \ldots, x_{k-1}\rangle$*, it holds that* $(x)_i = x_i$*.*

Proof. First notice that $\ell(x) = k$. Then apply the definitions to retrieve that

$$\begin{aligned}(x)_i &= (\langle x_0, \ldots, x_i, \ldots, x_{k-1}\rangle)_i\\ &= (\pi(k-1, \pi^k(x_0, \ldots, x_i \ldots, x_k) + 1))_i\\ &= \pi_{i+1}^k(\pi_2(\pi(k-1, \pi^k(x_0, \ldots, x_i, \ldots, x_k))))\\ &= \pi_{i+1}^k(\pi^k(x_0, \ldots, x_i, \ldots, x_k))\\ &= x_i.\end{aligned}$$

□

The functions not only work as intended, they are also both primitive recursive. The encoding acts as a bijection and hence $x = \langle (x)_0, \ldots, (x)_{\ell(x)-1}\rangle$. Another useful tool to have, is to replace elements of an encoded tuple.

Definition 3.8 *Let* x *be an encoded tuple, then the replacement function* $\mathrm{Put}(x, a, i)$ *puts an* a *at the* i*th place such that*

$$\mathrm{Put}(x, a, i) = \begin{cases}\langle (x)_0, \ldots, (x)_{i-1}, a, (x)_i, \ldots, x_{\ell(x)-1}\rangle & \text{if } 0 \leq i \leq \ell(x) - 1\\ x & \text{else}\end{cases}$$

It should be clear from the definition that this function is primitive recursive. For convenience a double replacement function is defined as $\mathrm{Put}(x, a, i, b, j) = \lambda xaibj.\, \mathrm{Put}(\mathrm{Put}(x, a, i), b, j)$, which is also primitive recursive by composition. This concludes the chapter about encodings, the introduced tools will be used in the next chapter to prove two theorems which have broad implications throughout mathematics in general and Hilbert's 10^{th} specifically.

4. Computability, Decidability & Enumerability

This chapter will finish part I. The most important results of elementary computability theory are stated here. Again, it is based on [19].

4.1 Equality of computability and recursiveness

The power of encodings is shown in this section. Here it is shown incrementally that the class of partial computable functions is exactly the class of partial recursive functions, thereby peeling of one layer of figure 2.2. Realise that an instruction of a program is just defined by two or three integers and hence a coding of a RM-instruction can be done as follows

$$r_i^+ \Rightarrow n \text{ gets code } \langle i, n\rangle$$
$$r_i^- \Rightarrow n, m \text{ gets code } \langle i, n, m\rangle.$$

A RM-program is a list of RM-structions $(p_1, \ldots, p_k)$, thus if for each p_j its encoding is $\hat{p_j}$ then the code of the RM-program is defined as $\langle \hat{p_1}, \ldots, \hat{p_k}\rangle$. All those encoded programs form a primitive recursive set Prog, so that

$$e \in \text{Prog} \Leftrightarrow \forall i < \ell(e).\big[((e)_i)_0 \geq 1 \wedge [\ell((e)_i) = 2 \vee \ell((e)_i) = 3]\big].$$

Which translates to "all commands must edit registers R_i where $i > 1$ and all instructions must either have two (r_i^+) or three (r_i^-) elements." The encoding of a whole computation can be defined similarly. A computation is a finite list of l-tuples $(r_1^i, \ldots, r_l^i)_{i=1}^K$, thus its encoding is given by

$$\langle\langle r_1^1, \ldots, r_l^1\rangle, \ldots, \langle r_1^K, \ldots, r_l^K\rangle\rangle.$$

This allows one to define the following predicate.

Definition 4.1 *The Kleene predicate $T(m, e, x, y)$ holds if and only if e is the code of a program P, where y is the code of the computation with P and input $\pi_1^m(x), \ldots, \pi_m^m(x)$.*

The following lemma is powerful, but its proof is not very elegant.

Lemma 4.1 *The Kleene predicate is a primitive recursive relation.*

Proof. The predicate $T(m, e, x, y)$ is a conjunction of the following statements:

- Code e must be a program: $\text{Prog}(e)$;
- A computation is never empty: $\ell(y) > 0$;
- Every tuple in the computation has the same size: $\forall i < \ell(y).[\ell((y)_i) = \ell((y)_0)]$;
- The tuples in the computation must be able to contain all the input variables: $\ell((y)_0) \geq m + 1$;

- The first tuple listed has an instruction pointer pointing to one on the first place and m elements which are the input of the computation:

$$((y)_0)_0 = 1 \wedge \forall i \leq m.[i \geq 1 \Rightarrow ((y)_0)_i = \pi_i^m(x)];$$

- The elements after the instruction pointer and the m input elements are zero: $\forall i < \ell((y)_0).[i > m \Rightarrow ((y)_0)_i = 0]$;
- Whenever a instruction pointer points to the STOP-instruction, this is the last step of the computation: $\forall i < \ell(y).[((y)_i)_0 = \ell(e) + 1 \Leftrightarrow i = \ell(y) - 1]$;
- If the at the jth step the instruction is of the form $r_i^+ \Rightarrow n$ and the registers contain $(k, r_1^j, \ldots, r_l^j)$, then at the next step the registers look like

$$(n, r_1^{j+1}, \ldots, r_{i-1}^{j+1}, r_i^j + 1, r_{i+1}^{j+1}, \ldots, r_l^j).$$

  This converts to:

$$\forall k < e, l < e, \forall i < \ell(y) - 1.\big[(e)_{((y)_i)_0} = \langle k, l\rangle \Rightarrow (y)_{i+1} = \mathrm{Put}((y)_i, l, 0, ((y)_i)_k + 1, k)\big];$$

- If the at the jth step the instruction is of the form $r_i^- \Rightarrow n, m$ and the registers contain $(k, r_1^j, \ldots, r_l^j)$, then at the next step the registers look like

$$(n, r_1^{j+1}, \ldots, r_{i-1}^{j+1}, r_i^j - 1, r_{i+1}^{j+1}, \ldots, r_l^j),$$

  if $r_i^j > 0$, otherwise the registers will look like $(m, r_1^{j+1}, \ldots r_l^j)$. This translates to: $\forall k, l, m < e, \forall i < \ell(y) - 1$ it holds that

$$\begin{aligned}&\forall k < e, l < e, m < e, \forall i < \ell(y) - 1.\big[(e)_{((y)_i)_0} = \langle k, l, m\rangle \Rightarrow\\ &[((y)_i)_k = 0 \wedge (y)_{i+1} = \mathrm{Put}((y)_i, m, 0)] \vee\\ &[((y)_i)_k = 0 \wedge (y)_{i+1} = \mathrm{Put}((y)_i, l, 0, ((y)_i)_k - 1, k)].\end{aligned}$$

Realise that each predicate is bounded and all their definitions are primitive recursive, because quantifiers, the logical relationships and the used functions are. Most of those have been proven in the previous sections and chapters, for further proofs see [3]. □

One extra function is needed.

Definition 4.2 *The output function U is defined such that whenever $T(m, e, x, y)$, $U(y)$ is the output of the computation.*

This means that, when $y = (r_1^i, \ldots, r_l^i)_{i=1}^K$ and $T(m, e, x, y)$ holds, the output function $U(y) = r_1^K$. Which is already a proof of its primitive recursiveness. Luckily this proof is a tiny bit shorter than that of lemma 4.1. This provides enough tooling to prove the following theorem.

Theorem 4.2 *A partial function is computable if and only if it is partial recursive.*

Proof. Suppose that f is a k-ary partial computable function, so that there exists a program P which computes f. Encode this program and call it e. Then it must be that

$$f(x_1, \ldots, x_k) \simeq U(\mu y.T(k, e, \pi^k(x_1, \ldots, x_k), y)).$$

But that means that every computable f is obtained by minimalization over the primitive recursive relation T (composed with primitive recursive function π^k). This shows that it is a partial recursive function! Recall that it was already established, via lemma 2.2 and lemma 1.7, that every partial recursive function is computable. □

Corollary 4.3 *Define a function $\Phi(m, e, x) \simeq U(\mu y.T(m, e, x, y))$. For any k-ary partial recursive function f there exists a number e (an encoded program) such that for all k-tuple $(x_1, \ldots, x_k)$ it holds that*

$$f(x_1, \ldots, x_k) \simeq \Phi(k, e, \pi^k(x_1, \ldots, x_k)).$$

In other words, there exists a universal function Φ.

Proof. Any partial recursive function is computable, thus there exists a program P to compute it, which can be encoded to e. This can then be applied to the function mentioned in theorem 4.2. □

For a partial function f, if it is written as $f = \lambda x_1 \ldots x_m.\Phi(m, e, \pi^m(x_1, \ldots, x_m))$, then e is called an index and it is written that $f = \varphi_e$. The notation $e \cdot (x_1, \ldots, x_m)$ is introduced to mean $\varphi_e(x_1, \ldots, x_m)$.

4.2 Decidability & Enumerability

It has been shown that subsets (and thus relations) of $\mathbb{N}^k$ and functions are intimately related. One example can be the set

$$\{x \mid \exists z \text{ such that } z^2 = x\},$$

which essentially describes the *problem* of finding perfect squares. In general a problem for a given set A is deciding whether an element is contained in A.

Definition 4.3 *Let $A \subset \mathbb{N}$, then A is called recursively enumerable if there is a partial recursive function ψ such that $A = \mathrm{dom}(\psi)$.*

This definition is equivalent to stating that A is recursively enumerable when there exists a partial recursive function ψ such that

$$\psi(x) = \begin{cases} 1 & \text{if } x \in A \\ \text{undefined} & \text{else} \end{cases}.$$

It is easy to verify that any recursive set is also recursively enumerable, by removing 0 from its domain. The relation between recursively enumerable and recursive is further clarified in the following lemma.

Lemma 4.4 *Let $R \subset \mathbb{N}^k$, then R is recursive if and only if R and $\overline{R}$ are recursively enumerable.*

Proof. If R is recursive, so is its complement and therefore are both recursively enumerable. In the other direction, write $R = \{\vec{x} \mid \exists y.\psi_R(\vec{x}, y)\}$ and $\overline{R} = \{\vec{x} \mid \exists y.\psi_{\overline{R}}(\vec{x}, y)\}$ for some recursive ψ_R and $\psi_{\overline{R}}$, which is possible, because both are assumed to be recursively enumerable. A k-ary function f can now be defined as $f(\vec{x}) \simeq \mu y.(\psi_R(\vec{x}, y) \vee \psi_{\overline{R}}(\vec{x}, y))$. See that $\mathrm{dom}(f) = R \cup \overline{R} = \mathbb{N}^k$ is total. □

If sets are recursive, then the set is called solvable or decidable. Now consider the halting set

$$\{(f, x) \mid \exists k, z \text{ such that } T(k, f, x, z)\},$$

which describes the problem of finding of all pairs (f, x) such that $f \cdot x$ is defined. Realise that this set is recursively enumerable. This is also referred to as the Halting problem, a quite famous problem. If the reader is familiar with the problem, the following theorem should not be a surprise.

Theorem 4.5 *The Halting problem is undecidable.*

Proof. Assume that the Halting problem is decidable. Then there is a characteristic function χ such that

$$\chi(f, x) = \begin{cases} 0 & \text{if } f \cdot x \text{ is defined} \\ 1 & \text{else} \end{cases}.$$

Define χ' as $\lambda x.\chi(x, x)$ and let g be any function such that $\mathrm{dom}(g) = \mathbb{N} \setminus \{0\}$. An example of such a function is $g(x) \simeq \mu y.xy > 1$. Now let f be the index of the function $\lambda x.g(\chi'(x))$. Then $f \cdot f$ is defined iff $\chi'(f) = 0$. However, $\chi'(f) = 0$ iff $g(\chi'(f))$ is undefined. But $g(\chi'(f))$ is equal to $f \cdot f$, which leads to a contradiction. This proves that our assumption was false. □

Corollary 4.6 *Not every recursively enumerable set is computable.*

Proof. Recall that the Halting set is recursively enumerable. However its complement is not, as it is impossible to compute whether an element is *not* in the set. □

Part II

Diophantine Set Theory

5. Diophantine Equations

This chapter is very loosely based on [16].

5.1 Introduction

The foundations in computability theory have been laid out. Recall the original problem this thesis set out to solve:

> *"Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers."*
>
> — **David Hilbert** (1900, [13][1])

Thus it is about time to exactly define Diophantine equations and dive into their properties.

Definition 5.1 *Let $f \in \mathbb{Z}[x_1, \ldots, x_m]$, then a Diophantine equation is an equation of the form*

$$f(x_1, \ldots, x_m) = 0.$$

The function f is then a Diophantine function.

Sometimes f in the above definition is referred to as a Diophantine polynomial. The equation above can also be referred to as a Diophantine equation in integers, so that the ring $\mathbb{Z}$ can be replaced for any commutative semiring. For example, it is also possible to consider Diophantine equations in naturals, where, in the definition, $\mathbb{Z}$ has been replaced by $\mathbb{N}$. For any Diophantine equation in integers an important question is to find values for $x_1, \ldots, x_m$ such that the equation holds. The following remark is made to broaden the ways of writing out Diophantine equations.

Remark 5.1 *Let $f, g \in \mathbb{Z}[x_1, \ldots, x_m]$, then*

$$f(x_1, \ldots, x_m) = g(x_1, \ldots, x_m)$$

can be transformed into a Diophantine equation.

Its proof should be trivial, assuming the reader is familiar with subtraction. Note that Diophantine functions are computable. Some famous examples of Diophantine equations are given.

Example 5.1 (Linear Diophantine Equation) *Fix a, b, then*

$$ax + by = 1$$

describes the linear Diophantine equation.

[1] Although the article is from 1902, the list was originally published in German in 1900.

Example 5.2 (Pell's Equation) *Fix integer a, then*

$$(x+2)^2 - ay^2 = 1$$

describes the Pell's equation.[2]

Example 5.3 (Sums of three cubes) *Fix an integer a, the equation*

$$x^3 + y^3 + z^3 = a$$

describes the sums of three cubes.[3]

In the previous part the most used number system was $\mathbb{N}$. One might be inclined to ask whether the requirement of f being an integer polynomial, in the definition of Diophantine equations, is necessary or that the naturals would suffice. They indeed suffice, which is shown in the following lemma.

Lemma 5.2 *Solving a Diophantine equation in integers is equivalent to solving a Diophantine equation in naturals.*

Proof. First assume that for any Diophantine in naturals it can be decided whether solutions exist. Let

$$f(x_1, \ldots, x_m) = 0$$

be an arbitrary Diophantine equation in integers. Any integer n can be written as a difference of two naturals, this naturally holds when $n \geq 0$ and when $n < 0$, one such way is to write $n = 0 - |n|$. Therefore, write each x_i as the difference of two naturals numbers, thus $x_i = y_i - z_i$, where $y_i, z_i \in \mathbb{N}$. Then

$$f(y_1 - z_1, \ldots, x_m - z_m) = 0$$

is a Diophantine equation in naturals and thus can be solved. Its solutions can be translated into solutions for the original equation.
Now assume that for any Diophantine in integers it can be decided whether solutions exist. Let

$$f(x_1, \ldots, x_m) = 0$$

be an arbitrary Diophantine equation in naturals. Recall the Langrange's four square theorem [15, pp. 280–284], i.e. any natural number can be expressed as the sum of four squares. Now write $x_i = a_i^2 + b_i^2 + c_i^2 + d_i^2$, where each $a_i, b_i, c_i, d_i \in \mathbb{Z}$. Then the solutions for

$$f(a_1^2 + b_1^2 + c_1^2 + d_1^2, \ldots, a_m^2 + b_m^2 + c_m^2 + d_m^2)$$

can be translated to solutions for the original equation. Showing all that needs to be shown. □

From now on, when speaking about Diophantine equations it is meant *in the naturals* unless explicitly stated otherwise.

5.2 Diophantine sets

A way to categorize Diophantine equations is to subdivide them into those that have solutions and those which have not.

[2] This Pell's equation has been slightly modified for educational purposes.
[3] This is not a very creative name, but that is how it is often referred to (see [4, 9, 17]).

Definition 5.2 *Let $f(a_1, \ldots, a_k, x_1, \ldots, x_m) = 0$ be a Diophantine equation with parameters $\vec{a} = a_1, \ldots, a_k$ and variables $\vec{x} = x_1, \ldots, x_m$. The Diophantine set D associated with f is defined as*

$$D = \{\vec{a} \in \mathbb{N}^k \mid \exists \vec{x} \in \mathbb{N}^m \text{ such that } f(\vec{a}, \vec{x}) = 0\}.$$

If $\vec{a}$ is contained in D then $f(\vec{a}, \vec{x})$ is called satisfiable under the parameters of $\vec{a}$. The equation $f(\vec{a}, \vec{x})$ is called the Diophantine representation of D.

The power of Diophantine sets lies in the relations such sets can describe, which will be further explored in section 5.3. The Diophantine sets of examples 5.1 to 5.3 can also be described as relations as follows.

Example 5.4 *The Diophantine set of the linear Diophantine equation is*

$$\{(a, b) \mid a \text{ and } b \text{ are coprime}\}.$$

The Diophantine set of the Pell's equation is

$$\{a \mid a \text{ is not a perfect square}\}.$$

Observe that example 5.3 is not mentioned in this example. This is because its Diophantine set is unknown, stated by the following two conjectures.

Conjecture 2 *The Diophantine set over the integers of the sums of three cubes equation is*

$$\{a \mid a \not\equiv \pm 4 \pmod 9\}.$$

This would seem fairly trivial, but realize that the smallest solution for $a = 42$ is

$$42 = (-80\,538\,738\,812\,075\,974)^3 + 80\,435\,758\,145\,817\,515^3 + 12\,602\,123\,297\,335\,631^3,$$

which was only found after a huge search using distributed computing [1].

Conjecture 3 *The set of integers which can be written as a sum of three cubes is incomputable.*

If conjecture 2 holds, this is not true and if conjecture 3 holds, the first does not have a proof. This little detour illustrates how hard even simple Diophantine equations can be. To connect with the previous part, the following lemma is stated.

Lemma 5.3 *All Diophantine sets are recursively enumerable.*

Proof. Let $f(\vec{a}, x_1, \ldots, x_m)$ be a Diophantine function and D be its Diophantine set, where $\vec{a}$ denotes the parameters and $x_1, \ldots, x_m$ denote the variables. Define the characteristic function

$$\psi(\vec{a}) \simeq \mu x.[f(\vec{a}, (x)_1, \ldots, (x)_m) = 0].$$

Then $\mathrm{dom}(\psi)$ is the set of values $\vec{a}$ for which a solution exists, which is exactly the Diophantine set D. □

The rest of this thesis is aimed at proving that the converse is also true: all recursively enumerable sets are Diophantine.

5.3 Diophantine relations

In example 5.4 it was shown that Diophantine sets can describe relations between natural numbers. This is precisely defined in the definition below.

Definition 5.3 *Let $S \subset \mathbb{N}^k$, then S is said to be a Diophantine relation when S is an exponential Diophantine set.*

A relation in itself is not really powerful. Therefore two operations are shown under which Diophantine relations are closed.

Lemma 5.4 *The union and intersection of Diophantine sets are also Diophantine.*

Proof. Let $f(\vec{x}) = 0$ and $g(\vec{x}) = 0$ be Diophantine equations. Their union is then

$$(f(\vec{x})g(\vec{x}) = 0,$$

because $fg = 0$ is satisfiable when eiter $f(\vec{x}) = 0$ or $g(\vec{x}) = 0$. By similar reasoning the intersection is

$$f(\vec{x})^2 + g(\vec{x})^2 = 0,$$

which is only satisfiable when both $f(\vec{x}) = 0$ and $g(\vec{x}) = 0$. □

Consequently, it is now possible to define Diophantine relations which are defined from multiple Diophantine properties. A few of those properties are now listed. To do so, the existential quantifier $\exists$ is used, which coincides with the definition of a Diophantine set. The items are listed from simple to complex, such that the simpler ones can be used to defined the more complex ones.

- $a \leq b$ can be represented by $\exists x.[a + x = b]$;
- $a < b$ can be represented by $\exists x.[a + x + 1 = b]$;
- Inequality $a \neq b$ holds iff $\exists x.[(a - b)^2 = x + 1]$;
- Divisibility $a \mid b$ is represented by $\exists x.[ax = b \text{ and } x \leq b]$;
- Integer division $c = a \div b$ is represented by $bc \leq a < (b + 1)c$. This integer division is also described with $c = \lfloor \frac{a}{b} \rfloor$;
- c is the remainder of integer division $a \div b$, noted with $c = a \,\%\, b$ and this is described with the Diophantine relation $c < b$ and $b \mid (a - c)$;
- Modular equality, $a \equiv b \pmod{c}$ is Diophantine because it can be written as $a \,\%\, c = b \,\%\, c$.

Equipped with these basic relations, it becomes possible to prove the two most important theorems of this thesis.

5.4 Exponential Diophantine sets

To show that all recursively enumerable sets are Diophantine, the concept of exponential Diophantine equations is introduced. By multiple theorems and lemmas it is then shown that the classes of Diophantine sets, Exponential Diophantine sets and recursively enumerable sets all coincide. A visual guide, also showing which lemma proves which property is shown in figure 5.1. A definition of exponential Diophantine equations is long overdue.

Definition 5.4 *Let $f(x_1, \ldots, x_m), g(x_1, \ldots, x_m)$ be exponential Diophantine functions: functions built from $x_1, \ldots, x_n$, natural numbers and addition, multiplication and exponentiation. Then an exponential Diophantine equation is an equation which be written as*

$$f(x_1, \ldots, x_m) = g(x_1, \ldots, x_m).$$

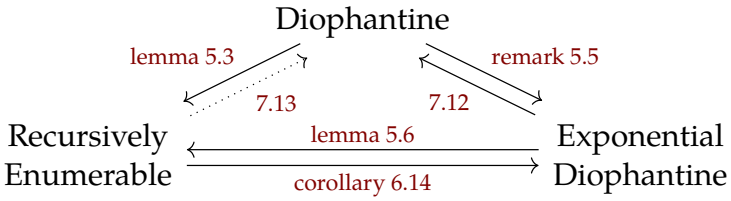


Figure 5.1: Relation of different proofs

Observe that subtraction is *not* allowed, in contrast to regular Diophantine equations. Furthermore, take note that exponential Diophantine functions are also computable, because exponentiation is computable (by primitive recursion of multiplication).

Example 5.5 *All classes of allowed possibilities are shown in*

$$3^x + 5y^z = 8^{zx^y} + 3y.$$

In a similar fashion to definition 5.2 exponential Diophantine sets can be defined.

Definition 5.5 *Let $f(\vec{a}, \vec{x}) = g(\vec{a}, \vec{x})$ be an exponential Diophantine equation with parameters $\vec{a}$ and variables $\vec{x}$. The exponential Diophantine set D associated with this equation is defined as the set of all $\vec{a}$ such that the exponential Diophantine equation is satisfiable.*

With the power of exponential Diophantine sets, some observations are made.

Remark 5.5 *Every Diophantine set is exponential Diophantine.*

This should not come as a surprise, as exponential Diophantine equations are just a *generalization* of Diophantine ones.

Lemma 5.6 *Every exponential Diophantine set recursively enumerable.*

Proof. Let $f(\vec{a}, \vec{x}) = g(\vec{a}, \vec{x})$ denote an exponential Diophantine equation in m variables and let D be its Diophantine set. Define the function

$$\psi(\vec{a}) \simeq \mu x.[f(\vec{a}, (x)_1, \ldots, (x)_m) = g(\vec{a}, (x)_1, \ldots, (x)_m],$$

which is computable. See that $\mathrm{dom}(\psi)$ is the set of $\vec{a}$ for which the exponential Diophantine equation has a solution, making it equal to D. □

The converse of both statements is much harder to prove and each deserves their own chapter. To prove those, some theory about how certain mathematical statements can be Diophantine is necessary. Thus the following remark is made.

Lemma 5.7 *The union and intersections of exponential Diophantine equations are also exponential Diophantine.*

Proof. Let $f_L(\vec{x}) = f_R(\vec{x})$ and $g_L(\vec{x}) = g_R(\vec{x})$ be exponential Diophantine equations. Their union is then

$$(f_L(\vec{x}) - f_R(\vec{x}))(g_L(\vec{x}) - g_R(\vec{x})) = 0.$$

It is exponential Diophantine because it can be rewritten to

$$f_L(\vec{x})g_L(\vec{x}) + f_R(\vec{x})g_R(\vec{x}) = f_L(\vec{x})g_R(\vec{x}) + f_R(\vec{x})g_L(\vec{x}).$$

By similar reasoning the intersection is

$$(f_L(\vec{x}) - f_R(\vec{x}))^2 + (g_L(\vec{x}) - g_R(\vec{x}))^2 = 0.$$

This is also exponential Diophantine because it can be rewritten to

$$f_L^2(\vec{x}) + f_R^2(\vec{x}) + g_L^2(\vec{x}) + g_R^2(\vec{x}) = 2f_L(\vec{x})g_R(\vec{x}) + 2g_L(\vec{x})f_R(\vec{x}). \qquad \square$$

6. Exponential Diophantineness of recursively enumerable sets

With the tools to describe relations between (exponential) Diophantine equations, it is possible to show that every recursively enumerable set is also exponential Diophantine. This proof was due to Julia Robinson, Hilary Putnam and Martin Davis [8], although this chapter is also based on the more modern works [7, 16]. In this thesis it is shown that Martin Davis's normal form is exponential Diophantine, the following theorem then does the rest.

Theorem 6.1 *For any recursively enumerable set A, there exists a Diophantine function f such that*

$$\vec{a} \in A \Leftrightarrow \exists z \forall y \leq z \exists x_1, \ldots, x_k.[f(\vec{a}, x_1, \ldots, x_m, y, z) = 0].$$

According to Robinson, Putnam and Davis this theorem is a very basic fact. Observe that if one were to get rid of the only universal quantifier, then one would prove that any Diophantine set is recursively enumerable. The proof of this theorem is in [6], which improves on the result of Kurt Gördel [12], who demonstrated that any recursively enumerable set can be represented by a Diophantine function f and an arbitrary number of universal quantifiers. In the next section a couple of functions is shown to be exponential Diophantine, so that in the second section it is shown that universal bound quantifiers are exponential Diophantine. This whole chapter, combined with its accompanying appendices, is somewhat of a 'tour de force', thus be prepared!

6.1 More exponential Diophantine relations

This section is all about the following theorem.

Theorem 6.2 *The functions*

$$\begin{aligned} f(n,k) &= \binom{n}{k}, \\ g(n) &= n! \quad \textit{and} \\ h(a,b,y) &= \prod_{k=1}^{y} (a+bk) \end{aligned}$$

are Diophantine.

This is proven using a multitude of lemmas, all of which are number theoretical facts tweaked for proving theorem 6.10. First the binomial coefficient function is shown to be a exponential Diophantine relation.

Lemma 6.3 *Let $0 < k \leq n$ and $u > 2^n$ be natural numbers, then*

$$\left\lfloor \frac{(u+1)^n}{u^k} \right\rfloor = \sum_{i=k}^{n} \binom{n}{i} u^{i-k}.$$

Proof. By Newton's binomial theorem, we can rewrite

$$\frac{(u+1)^n}{u^k} = \sum_{i=0}^{n} \binom{n}{i} u^{i-k}.$$

Split the sum in an integer part I and a small part S, so that

$$\sum_{i=0}^{n} \binom{n}{i} u^{i-k} = I + S = \sum_{i=k}^{n} \binom{n}{i} u^{i-k} + \sum_{i=0}^{k-1} \binom{n}{i} u^{i-k}.$$

Observe that indeed $I \in \mathbb{N}$. It is now only necessary to show that $S < 1$, such that $\left\lfloor \frac{(u+1)^n}{u^k} \right\rfloor = I$. This is shown by the inequalities:

$$\begin{aligned} S &< u^{-1} \sum_{i=0}^{k-1} \binom{n}{i} \\ &< u^{-1} \sum_{i=0}^{n} \binom{n}{i} \\ &= u^{-1} 2^n \\ &< 1. \end{aligned}$$

So the lemma holds. □

Corollary 6.4 *Let $0 < k \leq n$ and $u > 2^n$ be natural numbers, then*

$$\left\lfloor \frac{(u+1)^n}{u^k} \right\rfloor = \binom{n}{k} \pmod{u}.$$

Proof. This is an immediate consequence of lemma 6.3, as

$$\left\lfloor \frac{(u+1)^n}{u^k} \right\rfloor = \sum_{i=k}^{n} \binom{n}{i} u^{i-k} \equiv \binom{n}{k} \pmod{u}.$$ □

With this machinery in place the binomial coefficient function can now be tackled.

Lemma 6.5 *The function $f(n,k) = \binom{n}{k}$ is Diophantine.*

Proof. Let u be any natural number such that $u > 2^n$, so that the inequality

$$\binom{n}{k} \leq \sum_{i=0}^{n} \binom{n}{i} = 2^n < u$$

holds. Then, per corollary 6.4, the binomial coefficient $\binom{n}{k}$ is the lowest positive representation of $\left\lfloor \frac{(u+1)^n}{u^k} \right\rfloor$ modulo u. This means that when $x = \binom{n}{k}$, it can be represented by the following exponential Diophantine set:

$$\exists u, v, w.[v = 2^n \wedge u > v \wedge w = \left\lfloor \frac{(u+1)^n}{u^k} \right\rfloor \wedge z \equiv w \pmod{u} \wedge z < u].$$

To finish the proof, an argument is necessary to show that $\left\lfloor \frac{(u+1)^n}{u^k} \right\rfloor$ is Diophantine. If a positive integer $w = \left\lfloor \frac{(u+1)^n}{u^k} \right\rfloor$, then

$$\exists x, y, t.[t = u + 1 \wedge x = t^n \wedge y = u^k \wedge w = x \div y].$$

As all listed properties are exponential Diophantine, the lemma holds. □

To show that $\binom{n}{k}$ was Diophantine, an identity was proven from which it could be derived that it is exponential Diophantine. This will be done analogously for the factorial function.

Lemma 6.6 *Let r be such that $r > (2x)^{x+1}$, then*

$$x! = \left\lfloor r^x \Big/ \binom{r}{x} \right\rfloor.$$

The proof of this lemma is listed in appendix C.1. This identity allows us to show that the factorial operator is exponential Diophantine.

Lemma 6.7 *The function $g(n) = n!$ is exponential Diophantine.*

Proof. The number $m = n!$ can be represented by the exponential Diophantine set

$$\exists r, s, t, u, v.[s = 2x + 1 \wedge t = x + 1 \wedge r = s^t \wedge u = r^n \wedge v = \binom{r}{n} \wedge m = u \div v]. \quad \square$$

In a similar fashion an identity is first shown.

Lemma 6.8 *Let $bq \equiv a \pmod M$ hold, where all are natural numbers. Then*

$$\prod_{k=1}^{y}(a + bk) \equiv b^y y! \binom{q+y}{y} \pmod M.$$

Proof. Simply expand the $\binom{q+y}{y}$ term and simplify.

$$\begin{aligned} b^y y! \binom{q+y}{y} &= b^y (q+y)(q+y-1)\dots(q+1) \\ &= (bq+y)(bq+b(y-1))\dots(bq+b) \\ &\equiv (a+y)(a+b(y-1))\dots(a+b) \pmod M \\ &\equiv \prod_{k=1}^{y}(a+bk) \pmod M. \end{aligned}$$

□

Using this property, we can show that the product is exponential Diophantine.

Lemma 6.9 *The function $h(a, b, y) = \prod_{k=1}^{y}(a + bk)$ is exponential Diophantine.*

Proof. First let $M = b(a + by)^y + 1$ so that $\gcd(M, b) = 1$ and $M > \prod_{k=1}^{y}(a + bk)$. Observe that there is a solution for the congruence $bq \equiv a \pmod M$. Now $\prod_{k=1}^{y}(a + bk)$ is the lowest positive representation modulo M of $b^y y! \binom{q+y}{y}$. Which means

$z = \prod_{k=1}^{y}$ can be represented by the existential quantification:

$$\begin{aligned}\exists M, p, q, r, s, t, u, v, w, x.[r = a + by \wedge s = r^y \wedge M = bs + 1 \wedge \\ bq \,\%\, M = a \wedge u = b^y \wedge v = y! \wedge z < M \wedge \\ w = q + y \wedge x = \binom{w}{y} \wedge z \,\%\, M = uvx].\end{aligned}$$ □

This concludes the, somewhat intense, legwork which is necessary to show that bounded quantification is exponential Diophantine.

6.2 Bounded quantification

So far all our coding of relations has been done using the conjunction, disjunction and the existential quantification. There are however more quantifications, this thesis is concerned with the bounded versions of both the existential and universal quantifications. A proper definition will now be given.

Definition 6.1 *Let $\exists$ and $\forall$ be existential en universal quantifications. Let $P(x)$ denote a property of x, then the bounded quantifications are to be interpreted as follows:*

$$\exists y \leq x.P(y) \Leftrightarrow \exists y.[y \leq x \wedge P(y)]$$

is a bounded existential quantification and

$$\forall y \leq x.P(y) \Leftrightarrow \forall y.[y > x \vee P(y)]$$

is a bounded universal quantification.

The goal is to prove the following theorem.

Theorem 6.10 *Let f be a $m+1$ variable Diophantine function, then the sets*

$$E = \{(a_1, \ldots, a_n) \mid \exists k \leq y.[\exists z_1, \ldots, z_m.[f(a_1, \ldots, a_n, k, z_1, \ldots, z_m) = 0]]\}$$

and

$$A = \{(a_1, \ldots, a_n) \mid \forall k \leq y.[\exists z_1, \ldots, z_m.[f(a_1, \ldots, a_n, , z_1, \ldots, z_m) = 0]]\}$$

are exponential Diophantine.

This implies that the Davis normal form, theorem 6.1, is exponential Diophantine. The following lemma already proves half of the theorem and can be considered as low hanging fruit.

Lemma 6.11 *Bounded existential quantification is exponential Diophantine.*

Proof. This follows immediately from definition 6.1, because conjunction and $\leq$ are Diophantine. □

Universal bounded quantification is much harder to prove to be exponential Diophantine. It requires some setup. This is done by the following lemmas.

Lemma 6.12 *The expression*

$$P(\vec{a}, y) = \forall k \leq y.[\exists z_1, \ldots, z_m.[f(\vec{a}, k, z_1, \ldots, z_m) = 0]]$$

holds if and only if

$$Q(\vec{a}, y, B) = \forall k \leq y.\big[\exists z_1 \leq B, \ldots, z_m \leq B.[f(\vec{a}, k, z_1, \ldots, z_m) = 0]\big],$$

for some B, where f is a $m+1$ variable Diophantine function.

Proof. Observe that $Q(\vec{a}, y, B)$ immediately implies $P(\vec{a}, y)$. Now assume that $P(\vec{a}, y)$ holds, then define $z_1, \ldots, z_m$ to be those values making $P(\vec{a}, y)$ true. Let

$$B = \max\{z_1, \ldots, z_m\}$$

and see that this B suffices for $Q(\vec{a}, y, B)$ to hold. □

Similar to the previous section, a number theoretical identity is proven.

Lemma 6.13 *Let f and g be Diophantine functions with $m+3$ and 2 variables respectively. Moreover let g be such that $g(x, y) \geq y$, and*

$$\forall k \leq y, z_1 \leq y, \ldots, z_m \leq y.\big[|f(\vec{a}, x, y, k, z_1, \ldots, z_m)| \leq g(x, y)\big].$$

Then

$$\forall k \leq y.\big[\exists z_1 \leq y, \ldots, z_m \leq y.\,[f(\vec{a}, x, y, k, z_1, \ldots, z_m) = 0]\,\big]$$

holds if and only if

$$\exists c, t, b_1, \ldots, b_m.\big[t = g(x, y)! \wedge 1 + ct = \prod_{k \leq y} 1 + kt \wedge 1 + ct \mid f(\vec{a}, x, y, c, b_1, \ldots, b_m) \wedge \forall i \leq m.[1 + ct \mid \prod_{j \leq y} b_i - j]\big].$$

Note that the last statement is a finite conjunction of statements.

This identity establishes a bijection between $Q(\vec{a}, y)$ and an expression which does not contain any universal bounded quantifiers but instead consists out of a conjunction of properties which were shown to be exponential Diophantine. A proof of it is shown in [8, Lemma 3, pp. 433–435]. Now the theorem can be proven.

Proof of theorem 6.10. First construct a Diophantine function g such that both required properties of lemma 6.13 hold. Let f be a Diophantine polynomial in $m+3$ variables. Assume WLG its degree is $n > 0$, then it can be written as

$$f(z_1, \ldots, z_{m+3}) = \sum_{i_1 + \cdots + i_{m+3} \leq n} a_{i_1, \ldots, i_{m+3}} z_1^{i_1} \ldots z_{m+3}^{i_{m+3}}.$$

Define c as

$$c = \sum_{i_1 + \cdots + i_{m+3} \leq n} |a_{i_1, \ldots, i_{m+3}}|,$$

so that one can let $g(x,y) = cx^n y^n$. Observe that $g(x,y) \leq y$. Let x be arbitrary, fix y to be some natural number, consider $f(x, y, a_1, \ldots, a_{m+1})$ where each $a_i \leq y$, then

$$\begin{aligned}
|f(\vec{a}, x, y, z_1, \ldots, z_{m+1})| &= \left| \sum_{i_1+\cdots+i_{m+3} \leq n} a_{i_1,\ldots,i_{m+3}} x^{i_1} y^{i_2} z_1^{i_3} \ldots z_{m+1}^{i_{m+3}} \right| \\
&\leq \sum_{i_1+\cdots+i_{m+3} \leq n} |a_{i_1,\ldots,i_{m+3}}| x^{i_1} y^{i_2} z_1^{i_3} \ldots z_{m+1}^{i_{m+3}} \\
&\leq \sum_{i_1+\cdots+i_{m+3} \leq n} |a_{i_1,\ldots,i_{m+3}}| x^{i_1} y^{i_2+i_3+\cdots+m+3} \\
&\leq \sum_{i_1+\cdots+i_{m+3} \leq n} |a_{i_1,\ldots,i_{m+3}}| x^n y^n \\
&\leq cx^n y^n.
\end{aligned}$$

Having found such a g, the second step is to rewrite the universal bounded quantification. Recall from lemma 6.12 that

$$\forall k \leq y. \left[\exists z_1, \ldots, z_m. [f(\vec{a}, k, z_1, \ldots, z_m) = 0]\right]$$

can be considered as bounded on each variable so that lemma 6.13 can be invoked. Thus the universal bounded quantification from theorem 6.10 holds iff the properties from lemma 6.13 hold. It was already shown that factorial, the bounded product and the divisibility relation $|$ are exponential Diophantine. To see that also the relation $1 + ct \mid \prod_{j \leq y} b_i - j$ is exponential Diophantine, observe that it holds exactly when

$$a_i \leq y \text{ or } \exists u. \left[a_i > y \text{ and } (1+ct)u = \prod_{j \leq y} (a_i - y - 1 + j) \right]. \qquad \square$$

The immediate consequence is the second most important result of this thesis.

Corollary 6.14 *Every recursively enumerable set is exponential Diophantine.*

Proof. Combine theorem 6.1 with theorem 6.10. $\square$

Which is exactly the result this chapter set out to prove.

7. Undecidability of Hilbert's 10th

This chapter's sole goal is proving that exponentiation is Diophantine, that is,

$$a = b^c \Leftrightarrow \exists x_1, \ldots, x_m.[f(a, b, c, x_1, \ldots, x_m) = 0],$$

for some Diophantine polynomial f. Luckily, by now, some tooling has already been established. Sadly, this is not enough. Before the proof of Matiyasevich can be wholly understood, some extra understanding about recurrence relations needs to be set up. Recurrence relations are namely a way to describe exponential growth. If recurrence relations can be fully described in Diophantine relations, it is consequently itself Diophantine. All theory of this chapter is based on [16].

7.1 Properties of recurrence relations

The second-order recurrence relations this thesis is concerned with shall be denoted with $\alpha_b(n)$ where $b \geq 2$, $a_b(0) = 0$, $\alpha_b(1) = 1$ and for all $n \geq 2$

$$\alpha_b(n) = b\alpha_b(n-1) - \alpha_b(n-2).$$

Observe that

1) The sequence $\alpha_b(n)$ is monotonically increasing thus for all n it holds that $n \leq \alpha_b(n)$;

2) When $b = 2$ the sequence $\alpha_b(n)$ grows linearly;

3) Whenever $b > 2$ the sequence grows exponentially, to be more precise:

$$(b-1)^n \leq \alpha_b(n+1) \leq b^n \tag{7.1}$$

From now on whenever $\alpha(n)$ is written it is meant to represent $\alpha_b(n)$. Subscripts will only be written whenever not writing it causes ambiguity. Assuming that $\alpha(-1) = -1$, it becomes possible to get the values of $\alpha(n)$ via a first-order matrix recurrence relation. This can be done by writing

$$\Lambda(n) = \begin{pmatrix} \alpha(n+1) & -\alpha(n) \\ \alpha(n) & -\alpha(n-1) \end{pmatrix},$$

so that

$$\Lambda(n+1) = \Lambda(n)\Gamma \quad \text{where} \quad \Gamma = \begin{pmatrix} b & -1 \\ 1 & 0 \end{pmatrix}.$$

This holds because $\Lambda(0) = I_2$ and using the definitions of $\alpha(n)$ the matrix $\Lambda(n)\Gamma$ can be written as

$$\begin{aligned}\Lambda(n)\Gamma &= \begin{pmatrix}\alpha(n+1) & -\alpha(n)\\ \alpha(n) & -\alpha(n-1)\end{pmatrix}\begin{pmatrix}b & -1\\ 1 & 0\end{pmatrix}\\ &= \begin{pmatrix}b\alpha(n+1)-\alpha(n) & -\alpha(n+1)\\ b\alpha(n)-\alpha(n-1) & -\alpha(n)\end{pmatrix}\\ &= \begin{pmatrix}\alpha(n+2) & -\alpha(n+1)\\ \alpha(n+1) & -\alpha(n)\end{pmatrix}\\ &= \Lambda(n+1).\end{aligned}$$

This has the immediate consequence that $\Lambda(n) = \Gamma^n$. Another useful identity is obtained by writing

$$\begin{aligned}\alpha(n)\Gamma - \alpha(n-1)I_2 &= \begin{pmatrix}b\alpha(n) & -\alpha(n)\\ \alpha(n) & 0\end{pmatrix} - \begin{pmatrix}\alpha(n-1) & 0\\ 0 & \alpha(n-1)\end{pmatrix}\\ &= \begin{pmatrix}b\alpha(n)-\alpha(n-1) & -\alpha(n)\\ \alpha(n) & -\alpha(n-1)\end{pmatrix}\end{aligned}$$

So that

$$\Lambda(n) = \alpha(n)\Gamma - \alpha(n-1)I_2 \tag{7.2}$$

7.1.1 The characteristic polynomial of $\Lambda(n)$

As $\det(\Gamma) = 1$, all higher powers of Γ will also have determinant equal to one. Thus for the characteristic polynomial it holds that

$$\begin{aligned}\alpha^2(n) - \alpha(n+1)\alpha(n-1) =\\ \alpha^2(n) - (b\alpha(n) - \alpha(n-1))\alpha(n-1) =\\ \alpha^2(n) + \alpha^2(n-1) - b\alpha(n)\alpha(n-1) = 1.\end{aligned}$$

Considering this equation modulo $\alpha(n)$, one retrieves that

$$\alpha^2(n+1) \equiv 1 \pmod{\alpha(n)},$$

meaning that consecutive values of α are coprime. Realize that this holds for all n and thus, by substituting n for $n+1$, also that

$$\alpha^2(n+1) + \alpha^2(n) - b\alpha(n+1)\alpha(n) = 1.$$

The converse also holds.

Lemma 7.1 *The Diophantine equation*

$$x^2 - bxy + y^2 = 1$$

holds exactly when

$$x = \alpha(m+1), y = \alpha(m) \quad \textit{or} \quad x = \alpha(m), y = \alpha(m+1),$$

for some non-negative m.

Proof. WLG, assume that $y < x$. This will be proven by induction, where the base case is $y = 0$, for which the Diophantine equation holds if $x = 0$, which holds exactly when $m = 0$. Let $y > 0$ be such that the Diophantine equation holds, then, by our assumption that $y < x$,

$$by - x = \frac{y^2 - 1}{x} \geq 0.$$

Also realize that

$$\frac{y^2 - 1}{x} < \frac{y^2}{x} < y.$$

Define the variables $\hat{x} = y$ and $\hat{y} = by - x$, so that

$$\begin{aligned} \hat{x}^2 - b\hat{x}\hat{y} + \hat{y}^2 &= y^2 - by(by - x) + (by - x)^2 \\ &= x^2 - bxy + y^2 \\ &= 1. \end{aligned}$$

I.e. $\hat{x}$ and $\hat{y}$ also give a solution to the Diophantine equation. See that $\hat{y} < y = \hat{x}$, thus the IH can be applied, meaning that

$$\hat{x} = \alpha(\hat{m} + 1) \quad \text{and} \quad \hat{y} = \alpha(\hat{m}).$$

Which proves the lemma because, letting $m = \hat{m} + 1$, it follows that

$$x = b\hat{x} - \hat{y} = \alpha(m + 1) \quad \text{and} \quad y = \hat{x} = \alpha(m). \qquad \square$$

7.1.2 Divisibility & congruence properties

To further describe a recurrence relation in Diophantine properties, some divisibility properties need to be established. This is useful because divisibility is a Diophantine property.

Lemma 7.2 *For any positive k it holds that*

$$\alpha(k) \mid \alpha(m) \quad \textit{iff} \quad k \mid m.$$

Proof. A useful identity is established before the bi-directional implication is proven. Realize that $m \geq k$ and thus m can be written as $m = qk + r$, where $0 \leq r < k$. Recall additionally that $\alpha(k)$ and $\alpha(m)$ are elements of the matrices $\Lambda(k)$ and $\Lambda(m)$ respectively. Use these facts to rewrite $\Lambda(m)$ in the following steps:

$$\begin{aligned} \Lambda(m) &= \Gamma^m \\ &= \Gamma^{qk+r} \\ &= \Gamma^r \left(\Gamma^k\right)^q \\ &= \Lambda(r)\Lambda^q(k) \\ &= \begin{pmatrix} \alpha(r+1) & -\alpha(r) \\ \alpha(r) & -\alpha(r-1) \end{pmatrix} \begin{pmatrix} \alpha(k+1) & -\alpha(k) \\ \alpha(k) & -\alpha(k-1) \end{pmatrix}^q \\ &\equiv \begin{pmatrix} \alpha(r+1) & -\alpha(r) \\ \alpha(r) & -\alpha(r-1) \end{pmatrix} \begin{pmatrix} \alpha(k+1) & 0 \\ 0 & -\alpha(k-1) \end{pmatrix}^q \pmod{\alpha(k)}. \end{aligned}$$

Which means that

$$[\Lambda(m)]_{2,1} = \alpha(m) \equiv \alpha(r)\alpha^q(k+1) \pmod{\alpha(k)}, \tag{7.3}$$

the identity necessary for the rest of this lemma. Now the $\Rightarrow$ part is proven, thus $\alpha(k) \mid \alpha(m)$ holds. By equation 7.3 it then follows that $\alpha(k) \mid \alpha(r)\alpha^q(k+1)$. Earlier it was shown that consecutive values of α are coprime, hence $\alpha(k) \mid \alpha(r)$. From the monotonicity of α and our assumption it follows that $\alpha(r) < \alpha(k)$. This can holy hold if $r = 0$ which means that $k \mid m$. For the $\Leftarrow$ part it is assumed that $k \mid m$, and thus $r = 0$ and consequently $\alpha(r) = 0$, making $\alpha(m) \equiv 0 \pmod{\alpha(k)}$, proving the lemma. □

A second lemma about division is as follows.

Lemma 7.3 *For any positive k it holds that*

$$\alpha^2(k) \mid \alpha(m) \quad \textit{iff} \quad k\alpha(k) \mid m.$$

Proof. By lemma 7.2 it is necessary that $m = qk$ for some q. Using that fact it becomes possible to rewrite $A(m)$, by the following steps:

$$\begin{aligned}
\Lambda(m) &= \Gamma^{qk} \\
&= \Lambda^q(k) \\
&= (\alpha(k)\Gamma - \alpha(k-1)I_2)^q \\
&= \sum_{i=0}^{q} (-1)^{q-i} \binom{q}{i} \alpha^i(k)\alpha^{q-i}(k-1)\Gamma^i \\
&\equiv (-1)^q\alpha^q(k-1)I_2 + (-1)^{q-1}q\alpha(k)\alpha^{q-1}(k-1)\Gamma \pmod{\alpha^2(k)}
\end{aligned}$$

Observe that equation 7.2 and the binomial theorem are used. Consequently it also holds that

$$[\Lambda(m)]_{2,1} = \alpha(m) = (-1)^{q-1}q\alpha(k)\alpha^{q-1}(k-1)$$

Now, for $\Leftarrow$ it is assumed that $k\alpha(k) \mid m$, which means that $\alpha(k) \mid q$. This immediately implies that $\alpha(m) \equiv 0 \pmod{\alpha^2(k)}$, proving half of the bi-implication. For $\Rightarrow$ the above equation implies that $\alpha(k) \mid q\alpha^{q-1}(k-1)$. As consecutive values of α are coprime it must be that $\alpha(k) \mid q$ and thus $k\alpha(k) \mid kq = m$, proving the second part. □

Also some congruence properties are necessary, filling our backpack with useful proof tools.

Lemma 7.4 *Whenever $b_1 \equiv b_2 \pmod{q}$ it holds that $\alpha_{b_1} \equiv \alpha_{b_2} \pmod{q}$.*

Proof. This follows immediately by applying induction on the definition of α_b. □

Remark 7.5 *Because $b \equiv 2 \pmod{b-2}$ and α_2 is linear, it thus holds that*

$$\alpha_b(n) \equiv \alpha_2(n) = n \pmod{b-2}.$$

One last property will suffice to show the Diophantiness of α.

Lemma 7.6 *If $n = 2lm \pm j$, then*

$$\alpha(n) \equiv \pm\alpha(j) \pmod{v},$$

where $v = \alpha(m+1) - \alpha(m-1)$.

The $\pm$ signs need not to coincide.

Proof. The matrix notation is used again to obtain a certain identity. See that

$$\begin{aligned}\Lambda(n) &= \Gamma^n \\ &= \Gamma^{2lm\pm j} \\ &= \left[(\Gamma^m)^2\right]^l \Gamma^{\pm j} \\ &= \left(\Lambda^2(m)\right)^l \Lambda^{\pm 1}(j).\end{aligned}$$

Now for $\Lambda(m)$ it holds that

$$\Lambda(m) = \begin{pmatrix} \alpha(m+1) & -\alpha(m) \\ \alpha(m) & -\alpha(m-1) \end{pmatrix} \equiv - \begin{pmatrix} -\alpha(m-1) & \alpha(m) \\ -\alpha(m) & \alpha(m+1) \end{pmatrix} \pmod{v}. \quad (7.4)$$

The most right hand matrix is the inverse of $\Lambda(m)$:

$$\begin{aligned}&\begin{pmatrix} \alpha(m+1) & -\alpha(m) \\ \alpha(m) & -\alpha(m-1) \end{pmatrix} \begin{pmatrix} -\alpha(m-1) & \alpha(m) \\ -\alpha(m) & \alpha(m+1) \end{pmatrix} \\ =&\begin{pmatrix} \alpha^2(m) - \alpha(m+1)\alpha(m-1) & 0 \\ 0 & \alpha^2(m) - \alpha(m+1)\alpha(m-1) \end{pmatrix} \\ =&I_2.\end{aligned}$$

See that the property of the characteristic polynomial was used. By this result and equation 7.4, it can be shown that

$$\Lambda^2(m) \equiv -I_2 \pmod{b},$$

and thus

$$\Lambda(n) \equiv \pm\Lambda^{\pm 1}(j).$$

The $\pm$ sign before Λ is determined by whether l is odd or not, therefore any combination of plusses and minuses in the exponent and before Λ is possible. As a last step, see that

$$[\Lambda(n)]_{2,1} = \alpha(n) \equiv \pm\alpha(j) \pmod{v}.$$ □

This concludes all the lemmas necessary to define exponentiation as a Diophantine relation.

7.2 The sequence α is Diophantine

The sequence α has an exponential nature, therefore it is shown it is Diophantine, such that it can be used to show that exponentiation is Diophantine. Three numbers a, b and c are fixed such that

$$b > 3 \quad \text{and} \quad a = \alpha_b(c). \quad (7.5)$$

It is now claimed that these properties holds iff $\exists r, s, t, u, v, w, x, y$ such that all of the following conditions are true:

i) $3 < b$;

ii) $u^2 - but + t^2 = 1$;

iii) $s^2 - bsr + r^2 = 1$;

iv) $r < s$;

v) $u^2 \mid s$;

vi) $v = bs - 2r$;

vii) $w \equiv b \pmod{v}$;

viii) $w \equiv 2 \pmod{u}$;

ix) $w > 2$;

x) $x^2 - wxy + y^2 = 1$;

xi) $u > 2a$;

xii) $v > 2a$;

xiii) $a \equiv x \pmod{v}$;

xiv) $u > 2c$;

xv) $c \equiv x \pmod{u}$.

This is done by a sufficiency and a necessity lemma.

Lemma 7.7 *The conjunction of enumerated statements above contains sufficient statements to describe the sequence α.*

Proof. It was shown in lemma 7.1 that item i) and item ii) imply that $u = \alpha_b(k)$, for some natural k. Similarly $s = \alpha_b(m)$ and $r = \alpha_b(m-1)$, for some positive m, by items i), iii) and iv). Apply this reasoning again using items ix) and x) to conclude that $x = \alpha_w(n)$, for some n. Write for some $j \leq m$ that $n = 2lm \pm j$. Invoke lemma 7.3 in conjunction with item v) on the equalities of u and s to show that $u \mid m$. From item vi) and the definition of $\alpha_b(m)$ it follows that $v = \alpha_b(m+1) - \alpha_b(m-1)$. Items vii) and xiii), the properties of x, lemma 7.4 and lemma 7.6 show that

$$a \equiv x \equiv \alpha_b(w) \equiv \alpha_b(n) \equiv \pm\alpha_b(j) \pmod{v}.$$

From the properties of b, j, v, the definition of the sequence $\alpha_b(n)$ and the fact that α_b is monotonous increasing, the statement

$$2\alpha_b(j) \leq 2\alpha_b(m) \leq (b-2)\alpha_b(m) < b\alpha_b(m) - 2\alpha_b(m-1) = v$$

holds. But by item xi) and the fact that $a \equiv \pm\alpha_b(j) \pmod{v}$ it must be that $a = \alpha_b(j)$. Now it only needs to be shown that j equals c. First a property on c is derived by combining items ix) and xv) with remark 7.5 to retrieve that

$$c \equiv x \equiv \alpha_w(n) \equiv n \pmod{u}.$$

The consequence of the monotonicity of α_b can be combined with item xi) so that

$$2j \leq 2\alpha_b(j) = 2a < u,$$

which implies that, given item xiv),

$$c = j.$$

□

The necessity is equally necessary[1].

Lemma 7.8 *The conjunction of enumerated statements above contains necessary statements to describe the sequence α.*

Meaning that if equation 7.5 holds for a, b and c then there exists numbers s, r, u, t, v, w such that they satisfy all the properties listed. A proof is omitted here, but can be found in [16].

Theorem 7.9 *The recurrence relation α is Diophantine.*

Proof. Observe that by lemma 7.7 and lemma 7.8 the conjunction of statements is necessary and sufficient. Furthermore, each of them is Diophantine, because the modularity relation, $<$-relation and the divisibility relation are Diophantine. □

7.3 Exponentiation is Diophantine

This thesis is reaching its apotheosis, which is indeed suggested by the title. A little more setup is still needed, but the exponentiation is already looking at us from the statement of the lemma.

Lemma 7.10 *Let λ be an eigenvalue of the matrix Γ such that $\lambda \equiv q \pmod m$, then*

$$q\alpha(r) - \alpha(r-1) \equiv q^r \pmod m.$$

Proof. See that the eigenvalue of Γ satisfies the characteristic equation $\lambda^2 - b\lambda - 1 = 0$. Thus, modulo m, it holds that $q^2 - bq + 1 \equiv 0$, which means that

$$q^2 \equiv bq - 1 \pmod m.$$

It is claimed that $\begin{pmatrix} q \\ 1 \end{pmatrix}$ is the eigenvector, modulo m, of Γ. This is shown by

$$\Gamma \begin{pmatrix} q \\ 1 \end{pmatrix} = \begin{pmatrix} bq - 1 \\ q \end{pmatrix} \equiv \begin{pmatrix} q^2 \\ q \end{pmatrix} \equiv q \begin{pmatrix} q \\ 1 \end{pmatrix} \pmod m.$$

Which means that

$$\begin{aligned} \Lambda(r) \begin{pmatrix} q \\ 1 \end{pmatrix} &= \Gamma^r \begin{pmatrix} q \\ 1 \end{pmatrix} \\ &\equiv q^r \begin{pmatrix} q \\ 1 \end{pmatrix}. \end{aligned}$$

The lower row results in the oh so desired exponentiation. Thus most importantly

$$q\alpha_b(r) - \alpha_b(r-1) \equiv q^r \pmod m.$$

□

See that when $q^r < m$ the relation $p = q^r$ can be written into the properties

$$p < m \quad \text{and} \quad q\alpha_b(r) - \alpha_b(r-1) \equiv p \pmod m.$$

Theorem 7.11 *Exponentiation is Diophantine.*

[1] Pun intended

Proof. Assume one wants to encode $p = q^r$ as a Diophantine relation. Define $b = \alpha_{q+4}(r+1) + q^2 + 2$. Moreover let λ be an eigenvalue of Γ and define $m = bq - q^2 - 1$. Then $q^2 - bq + 1 \equiv 0 \pmod{m}$, meaning that q acts as an eigenvalue modulo m, thus $\lambda \equiv q \pmod{m}$. Now invoke lemma 7.10 so that $q\alpha(r) - \alpha(r-1) \equiv q^r$. Use equation 7.1, assuming that $q > 0$, to show that $q^r < m$:

$$\begin{aligned} m &= bq - q^2 - 1 \\ &= (\alpha_{q+4}(r+1) + q^2 + 2)q - q^2 - 1 \\ &\geq q(q+3)^r + q^3 + 2q - q^2 - 1 \\ &\geq q(q+3)^r \\ &> q^r. \end{aligned}$$

Assuming that $0^0 = 1$[2] $p = q^r$ holds exactly when the following Diophantine properties hold:

$$\begin{aligned} &(q = 0 \wedge r = 0 \wedge p = 1) \vee \\ &(q = r \wedge r > 0 \wedge p = 0) \vee \\ \exists b, m. \big[&b = \alpha_{q+4}(r+1) + q^2 + 2 \wedge m = bq - q^2 - 1 \wedge \\ &p < m \wedge p \equiv q\alpha_b(r) - (b\alpha_b(r) - \alpha_b(r+1))\big]. \end{aligned}$$ □

The theorem has the following implication.

Corollary 7.12 *Any exponential Diophantine set is Diophantine.*

And the more important one.

Corollary 7.13 *All recursive enumerable sets are Diophantine.*

Which had the consequence that Hilbert's 10$^{\text{th}}$ Problem did not have a solution.

Theorem 7.14 *There is no general algorithm which can determine the solutions for an arbitrary Diophantine equation.*

Proof. Assume there exists a general algorithm. Then it can also determine whether there is a solution and thus it can compute Diophantine sets. As any recursively enumerable set is Diophantine, any recursively enumerable set is in bijection with a Diophantine set. By corollary 4.6 some recursively enumerable sets are incomputable. A contradiction! There must be Diophantine sets which are incomputable. Ergo, there cannot be a general algorithm determining the solutions for arbitrary Diophantine equations. □

[2] a valid assumption when working over $\mathbb{N}$.

Part III

Appendices

A. Computability proofs

A.1 Outputting input is computable

Before starting the proof, the following convention is being used. Whenever a step in a computation is described by something like $a\,\vec{b}\,\vec{c}\,d$, it is meant that there is a tuple $(a, b_1, \ldots, b_n, c_1, \ldots, c_m, d)$ in that computation step.

Proof. As f is computable, it has an associated program Q, moreover, let $\vec{a} \in \mathrm{dom}(f)$. Assume Q uses all the registers up to R_l and that the computation of $f(a)$ with Q takes K steps, where the last tuple is $(n_K, f(\vec{a}), r_2^K, \ldots, r_l^K)$. Using the programs from example 1.1, the following program has the required output:

$$
\begin{array}{c}
\vec{a} \\
\Big\Vert C(1,l+1) \\
\vec{a}\,\vec{0}\,a_1 \\
\Big\Vert C(i,l+i) \\
\vdots \\
\Big\Vert C(k,l+k) \\
\vec{a}\,\vec{0}\,\vec{a} \\
\Big\Vert P \\
f(\vec{a})\,r_2^K\ \ldots\ r_l^K\,\vec{a} \\
\Big\Vert E(2) \\
f(\vec{a})\,0\,r_3^K\ \ldots\ r_l^K\,\vec{a} \\
\Big\Vert E(i) \\
\vdots \\
\Big\Vert E(l) \\
f(\vec{a})\,\vec{0}\,\vec{a} \\
\Big\Vert C(l+1,2) \\
f(\vec{a})\,a_1\,\vec{0}\,a_2\ \ldots\ a_k \\
\Big\Vert C(l+i,i+1) \\
\vdots \\
\Big\Vert C(l+n,n+1) \\
f(\vec{a})\,\vec{a}
\end{array}
$$

□

From now on, when reasoning about programs, the arrows which indicate simple combinations of the operations copying, emptying, adding and subtracting will not be labelled. Additionally, $\widetilde{P}$ will denote the modification of program P, as described in lemma 1.4.

A.2 Composition of computable functions

Proof. Assume that f is a k-ary function made from composing the $g_1, \ldots, g_l$, all k-ary computable functions and h, a l-ary partial function. Furthermore let P_i be the program computing g_i, Q the program computing h and $\vec{a} \in \mathrm{dom}(f)$. Now build a program as shown below.

$$
\begin{array}{c}
\vec{a} \\
\Big\Downarrow \widetilde{P_1} \\
g_1(\vec{a})\,\vec{a} \\
\Big\Downarrow \\
\vec{a}\,\vec{0}\,g_1(\vec{a}) \\
\Big\Downarrow \widetilde{P_2} \\
g_2(\vec{a})\,\vec{a}\,\vec{0}\,g_1(\vec{a}) \\
\Big\Downarrow \widetilde{P_i} \\
\vdots \\
\Big\Downarrow \widetilde{P_l} \\
g_l(\vec{a})\,\vec{a}\,\vec{0}\,g_1(\vec{a})\,\ldots\,g_{l-1}(\vec{a}) \\
\Big\Downarrow \\
g_1(\vec{a})\,\ldots\,g_l(\vec{a}) \\
\Big\Downarrow Q \\
h(g_1(\vec{a}), \ldots, g_l(\vec{a}))
\end{array}
$$

Ergo, f is computable. □

A.3 Primitive recursion is computable

Proof. Assume f to be defined from primitive recursion of computable functions g, k-ary, and h, $k+2$-ary. Some notation is introduced first. Realise that the instruction

1 $\quad r_i^- \Rightarrow n, m$

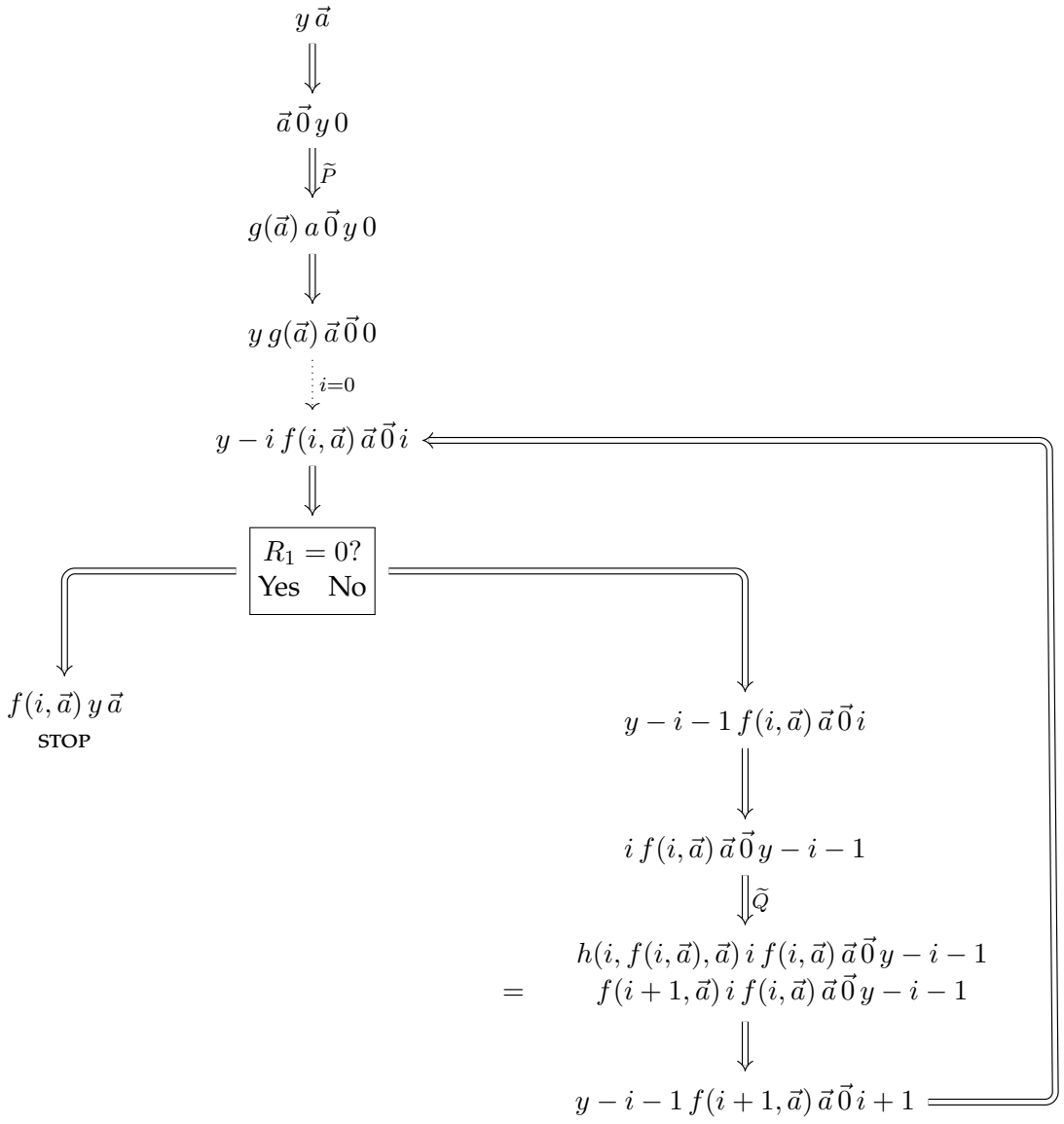
can be translated semantically into the simple conditional

```
if  r_i > 0 :
      move to instruction n
else:
      move to instruction m,
```

which will be depicted in a program diagram as follows

$$\boxed{\begin{array}{c} R_1 = 0? \\ \text{Yes} \quad \text{No} \end{array}}\ .$$

To complete the proof, assume that g is computable with program P and h is computable with program Q.

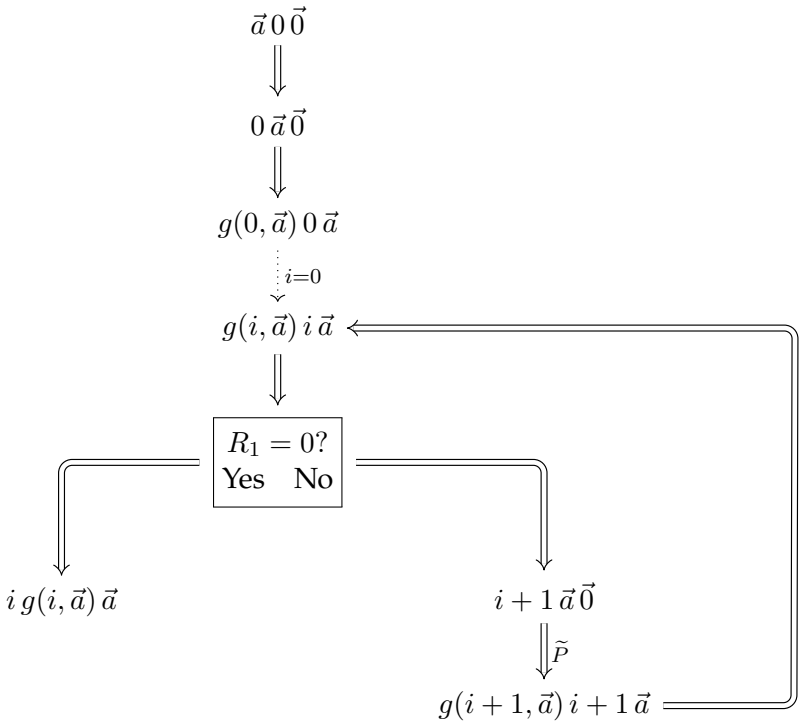


□

See that, in the definition of primitive recursion, $f(y, \vec{a})$ is defined *backwards* (by defining it using lower values than y. However, the calculation of $f(y, \vec{a})$ is done *forwards*. One needs the previously computed value, to calculate the next one.

A.4 Minimalization is computable

Proof. Let P be the program computing the $k+1$-ary function g, then the following program computes f:



□

B. Pairing function proofs

B.1 Bijectivity of the Cantor pairing function

Proof. For injectivity, a proof similar like [22] is given. Assume that $\pi(x, y) = \pi(z, w)$. The first thing to prove is that $x + y = z + w$. Assume WLG $x + y < z + w$ and substitute $a = x + y$ and $b = z + w - a$. See that both a and b are still natural numbers and that $b > 0$. Consequently the equality translates to:

$$\frac{a(a+1)}{2} + y = \frac{(b+a)(b+a+1)}{2} + w.$$

Rewrite the equation,

$$\begin{aligned} y - w &= \frac{(b+a)(b+a+1)}{2} - \frac{a(a+1)}{2} \\ &= \frac{b^2 + 2ab + b}{2} \\ &= ab + \frac{b(b+1)}{2} \\ &\geq a + 1, \end{aligned}$$

so that $y > w + a \geq a$. This leads to nonsense, because when a gets substituted back, the conclusion is that $y > a > x + y \geq y$. Therefore the assumption was false and $x + y = z + w$. Filling in $a = x + y$ results then in

$$\frac{a(a+1)}{2} + y = \frac{a(a+1)}{2} + w,$$

meaning that $y = w$ and consequently that $x = z$.
For surjectivity a construction is given, which is the author's interpretation from [14]. Assume there exists a $z = \pi(x, y)$ and define the following values:

$$\begin{aligned} a &= x + y \\ b &= \frac{a(a+1)}{2}, \end{aligned}$$

so that $z = b + y$. The next goal is to write a in terms of b, instead of otherwise. Using the quadratic formula on $\frac{a(a+1)}{2} - b = 0$, one gets that

$$a = \frac{\sqrt{8b+1} - 1}{2}.$$

Considered as a function $\mathbb{R} \to \mathbb{R}$, this is strictly increasing when $b \in [0, \infty)$. Therefore, when considering inequalities in terms of b, they still hold when rewritten in terms of a, which is precisely the next step. See that $b \geq z = b + y < b + (a + 1)$, from which

the RHS can be rewritten to

$$\begin{aligned} b + (a+1) &= \frac{a(a+1)}{2} + a + 1 \\ &= \frac{a^2 + 3a + 2}{2} \\ &= \frac{(a+1)^2 + (a+1)}{2}, \end{aligned}$$

which should seem familiar. Now this inequality means that, when rewriting a in terms of b,

$$a \leq \frac{\sqrt{8z+1} - 1}{2} < a + 1.$$

Consequently it must be that

$$a = \left\lfloor \frac{\sqrt{8z+1} - 1}{2} \right\rfloor.$$

In conclusion, given any $z \in \mathbb{N}$, it is possible to give $x, y \in \mathbb{N}$ such that $\pi(x, y) = z$, namely:

$$\begin{aligned} a &= \left\lfloor \frac{\sqrt{8z+1} - 1}{2} \right\rfloor \\ b &= \frac{a^2 + a}{2} \\ y &= z - b \\ x &= a - y. \end{aligned}$$

The desired result that the Cantor pairing function is a pairing function, is obtained. □

One small example will be given to show the inner workings of those functions.

Example B.1 *Let* $z = 6$*, then* $a = \lfloor \frac{\sqrt{48+1}-1}{2} \rfloor = \lfloor \frac{\sqrt{49}-1}{2} \rfloor = \lfloor \frac{6}{2} \rfloor = \lfloor 3 \rfloor = 3$ *and* $b = \frac{3^2+3}{2} = 6$*. Thus* $y = 0$ *and* $x = 3$*, which can be seen in figure 3.1, where* $(3, 0)$ *is the sixth tuple being listed. But it can also be calculated by* $\pi(3, 0) = \frac{3(3+1)}{2} = 6$*.*

B.2 Computability of the Cantor pairing and projection

Proof. The Cantor pairing function contains a division by two in its formulation. It has already been noted that the numerator is always even. Therefore division by two will never yield a remainder. Integer division (see [23]) will therefore suffice and can be defined as follows:

$$a \div b = \begin{cases} 0 & \text{if } a = 0 \text{ or } b = 0 \\ 1 + ((a \dot{-} b) \div b) & \text{else} \end{cases}.$$

This is primitive recursive since it is a composition of the C function and the primitive recursive functions Add and $\dot{-}$. WLG it is only necessary to prove that π_1 is primitive recursive (and hence computable). As it is the bounded minimalization, it is sufficient

to show that $\lambda y.[\exists y \leq z.\pi(x, y) = z]$ is primitive recursive. This is done by

$$\chi_{\exists y} = \operatorname{sgn}\left(\sum_{y \leq z} \chi_{\mathrm{eq}}(\pi(x, y), z)\right).$$

Realise that $\chi_{\exists y}$ is primitive recursive as it is the summation of a composition of primitive recursive functions. □

C. Proofs of multiple combinatorial identities

C.1 Factorial identity proof

Proof. Let r be any natural number such that $r > (2x)^{x+1}$. Recall that

$$\binom{r}{x} = \frac{r!}{x!(r-x)!}.$$

Use this to rewrite

$$\begin{aligned} r^x \Big/ \binom{r}{x} &= \frac{r^x x!(r-x)!}{r!} \\ &= \frac{r^x x!}{r(r-1)\dots(r-x+1)} \\ &= x!\left(\frac{r}{r}\frac{r}{r-1}\dots\frac{r}{r-x+1}\right) \\ &= \frac{x!}{\left(\left(1-\frac{1}{r}\right)\dots\left(1-\frac{x-1}{r}\right)\right)}. \end{aligned}$$

It now holds that

$$x! < r^x \Big/ \binom{r}{x} < \frac{x!}{\left(1-\frac{x}{r}\right)^x}.$$

By assumption it holds that $r > (2x)^{x+1}$, thus also that $r > 2x$, which means that the ratio $\frac{x}{r} < \frac{1}{2}$. Use this to see that

$$\begin{aligned} \frac{1}{1-\frac{x}{r}} &= 1 + \frac{x}{r} + \left(\frac{x}{r}\right)^2 + \dots \\ &= 1 + \frac{x}{r}\left(1 + \frac{x}{r} + \left(\frac{x}{r}\right)^2 + \dots\right) \\ &< 1 + \frac{x}{r}\left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right) \\ &= 1 + \frac{2x}{r}. \end{aligned}$$

Moreover it holds that, by using the Newton's binomial theorem again,

$$\begin{aligned} \left(\frac{1}{1-\frac{x}{r}}\right)^x &= \sum_{j=0}^{x}\binom{x}{j}\left(\frac{2x}{r}\right)^j \\ &< 1 + \frac{2x}{r}\sum_{j=1}^{x}\binom{x}{j} \\ &< 1 + \frac{2x}{r}\cdot 2^x. \end{aligned}$$

Using this in our first deduction, combined with our assumption about r and that $x! < x^x$, leads us to concluding that

$$\begin{aligned} r^x \Big/ \binom{r}{x} &< x! + x! \frac{2x}{r} \cdot 2^x \\ &< x! + \frac{2x \cdot x^x 2^x}{r} \\ &< x! + \frac{(2x)^{x+1}}{r} \\ &< x! + 1. \end{aligned}$$

The desired result is obtained. □

D. Index

E. Acknowledgements

This thesis has been mostly my own work. However, some people provided help which bettered my thesis. Those people deserve a mention. First and foremost, I thank Sunil Patel, for without him I would not have been able to write in such a beautiful style. Although I have edited the template in some points. Moreover, I want to thank Yichuan Shen which provided a very useful website to create commutative diagrams. I used this tool to create any diagram containing arrows. Lastly I want to thank the people who have read my thesis so that I was able to remove some critical mistakes, those are Chaja van Ansenwoude en Jaap van Oosten. Jaap van Oosten deserves a special mention, as he, being my supervisor, has let me mostly free in what I did, so that I was able to explore all the subjects I found interesting.

F. Bibliography

[1] Andrew R. Booker and Andrew V. Sutherland. "On a question of Mordell". In: *Proceedings of the National Academy of Sciences* 118.11 (2021). DOI: https://doi.org/10.1073/pnas.2022377118.

[2] George S. Boolos, John P. Burgess, and Richard C. Jeffrey. *Computability and Logic*. Cambridge: Cambridge Cambridge Univ. Press, 2010. ISBN: 978-0521701464.

[3] *Category: Primitive recurisve functions*. URL: https://proofwiki.org/wiki/Category:Primitive_Recursive_Functions.

[4] W. Conn and L.N. Vaserstein. "On sums of three integral cubes". In: *The Rademacher Legacy to Mathematics*. Ed. by George E. Andrews, David M. Bressoud, and L. Alayne Parson. Contemporary Mathematics. Providence: AMS, 1994. Chap. 21, pp. 285–294. ISBN: 978-0-8218-7757-9. DOI: http://dx.doi.org/10.1090/conm/166.

[5] Zoltán Csörnyei and Gergely Dévai. "An Introduction to the Lambda Calculus". In: *Central European Functional Programming School: Second Summer School*. Ed. by Zoltán Horváth et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. Chap. 3, pp. 87–111. ISBN: 978-3-540-88058-5. DOI: https://doi.org/10.1007/978-3-540-88059-2_3.

[6] Martin Davis. "Arithmetical Problems and Recursively Enumerable Predicates". In: *The Journal of Symbolic Logic* 18.1 (1953), pp. 33–41. DOI: https://doi.org/10.2307/2266325.

[7] Martin Davis. "Hilbert's Tenth Problem is Unsolvable". In: *The American Mathematical Monthly* 80.3 (1973), pp. 233–269. DOI: https://doi.org/10.2307/2318447.

[8] Martin Davis, Hilary Putnam, and Julia Robinson. "The Decision Problem for Exponential Diophantine Equations". In: *Annals of Mathematics* 74.3 (1961), pp. 425–436. DOI: https://doi.org/10.1090/S0025-5718-08-02168-6.

[9] Andreas-Stephan Elsenhans and Jahnel Jörg. "New sums of three cubes". In: *Mathematics of Computation* 78.266 (2009), pp. 1227–1230. DOI: https://doi.org/10.1090/S0025-5718-08-02168-6.

[10] William Ewald and Wilfried Sieg. "Lectures on the Infinite". German. In: *David Hilbert's Lectures on the Foundations of Arithmetic and Logic 1917-1933*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 655–785. ISBN: 978-3-540-69444-1. DOI: https://doi.org/10.1007/978-3-540-69444-1_4.

[11] Rudolf. Fueter and G. Pólya. "Rationale Abzählung der Gitterpunkte". German. In: *Vierteljahrsschrift der Naturforschenden* 68.3/4 (1923), pp. 380–386. URL: https://www.ngzh.ch/archiv/1923_68/68_3-4/68_14.pdf.

[12] Kurt Gödel. "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I". German. In: *Monatshefte für Mathematik und Physik* 38.1 (1931), pp. 173–198. DOI: https://doi.org/10.1007/BF01700692.

[13] David Hilbert. "Mathematical Problems". In: *Bulletin of the American Mathematical Society* 8.10 (1902), 437—479. DOI: https://doi.org/10.1090/S0002-9904-1902-00923-3.

[14] *Inverting the Cantor pairing function*. URL: https://en.wikipedia.org/wiki/Pairing_function#Inverting_the_Cantor_pairing_function (visited on 05/18/2021).

[15] K. Ireland and M. Rosen. *A Classical Introduction to Modern Number Theory*. 2nd ed. Graduate Texts in Mathematics. New-York: Springer-Verlag, 1990. ISBN: 978-1-4757-2103-4. DOI: https://doi.org/10.1007%2F978-1-4757-2103-4.

[16] Yuri Matiyasevich. *On Hilbert's Tenth Problem*. 2000. URL: https://mathtube.org/lecture/notes/hilberts-tenth-problem (visited on 09/21/2020).

[17] L. J. Mordell. "On Sums of Three Cubes". In: *Journal of the London Mathematical Society* s1-17.3 (1942), pp. 139–144. DOI: https://doi.org/10.1112/jlms/s1-17.3.139.

[18] John von Neumann. "First Draft of a Report on the EDVAC". In: *IEEE Annals of the History of Computing* 15.4 (1945), pp. 37–75. URL: https://doi.org/10.1109/85.238389.

[19] Jaap van Oosten. *Basic Computability Theory*. https://webspace.science.uu.nl/~ooste110/syllabi/compthmoeder.pdf. Written lecture notes. 1993.

[20] Steven Pigeon. *Pairing Function*. From MathWorld–A Wolfram Web Resource. URL: https://mathworld.wolfram.com/PairingFunction.html (visited on 05/29/2021).

[21] Arnold L. Rosenberg. "Efficient pairing functions – And why you should care". In: *International Journal of Foundations of Computer Science* 14.1 (2002), pp. 3–17. DOI: https://doi.org/10.1142/S012905410300156X.

[22] Brian M. Scott. *Inverting the Cantor pairing function*. Dec. 14, 2011. URL: https://math.stackexchange.com/a/91323/396012 (visited on 05/18/2021).

[23] Eric W. Weisstein. *Integer Division*. From MathWorld–A Wolfram Web Resource. URL: https://mathworld.wolfram.com/IntegerDivision.html (visited on 05/22/2021).