Semi-Countable Sets and their Application to Search Problems

Pieter Adriaans

ILLC, FNWI-IVI, SNE
University of Amsterdam,
Science Park 107
1098 XG Amsterdam,
The Netherlands. P.W.Adriaans@uva.nl

Abstract. We present the concept of the information efficiency of functions as a technique to understand the interaction between information and computation. Based on these results we identify a new class of objects that we call Semi-Countable Sets. As the name suggests these sets form a separate class of objects between countable and uncountable sets. In principle these objects are countable, but the information in the descriptions of the elements of the class grows faster than the information in the natural numbers that index them. Any characterization of the class in terms of natural numbers is fundamentally incomplete. Semi-countable sets define one-to-one injections into the set of natural numbers that can be computed in exponential time, but not in polynomial time. A characteristic semi-countable object is $\phi_\Sigma$ the set of all additions for all finite sets of natural numbers. The class $\phi_\Sigma$ codes the Subset Sum problem. This gives a natural and transparent analysis of the separation between the classes $P$ and $NP$.

Keywords: Philosophy of information, Information efficiency of functions, semi-countable sets, recursive functions, Elastic transformations over $\mathbb{N}^2$, $P$ vs. $NP$

1 Introduction

This paper develops some ideas that were presented in an elementary form in [5], where we argued that the most urgent problem of modern philosophy of information was our lack of understanding of the interaction between information and computation. For a deeper understanding of the philosophical backgrounds we refer to this publication.

1.1 Informal presentation of the main argument

Interaction between information and computation is a phenomenon that we are all familiar with from a cognitive point of view, but that until now has eluded mathematical conceptualization. Suppose we want to add a set of numbers $s_1 = \{3, 4, 6, 7\}$. We could compute $((3 + 4) + 6) + 7 = 20$, but most of us
would immediately see that is easier to compute \((3+7)+(4+6) = 10+10 = 20\). Such a trick is not available for the set \(s_2 = \{2, 5, 6, 7\}\). This example shows that, for some sets, the sequence of our computations influences the hardness of the problem. The observation that \(\Sigma_{i\in s_1} i = 20\) is, from this perspective, less surprising than the fact that \(\Sigma_{i\in s_2} i = 20\). From an information theoretical point of view this implies that the statement \(\Sigma_{i\in s_1} i = 20\) contains less information than the statement \(\Sigma_{i\in s_2} i = 20\). Since addition is commutative and associative we have no mathematical tools to explain this phenomenon in classical arithmetic.

What we need is a theory that helps us to distinguish the information aspects of different computational histories. Below we develop such a theory. The central concept is the notion of the information efficiency of a function as the balance between the information in the input and the information in the output. For addition this gives \(\delta(x + y) = \log(x + y) - \log x - \log y\). It is clear that this operation is not associative \(\delta((a + b) + c) \neq \delta(a + (b + c))\). The amount of possible computational histories for addition of a set of numbers is super exponential in its cardinality. This implies that the computational history of the way the output is computed is relevant for the amount of information it contains conditional to the input. In other words: even if we have the answer, we don’t know what we know, until we know how it is computed.

For some types of problems this means that, knowing the answer does not help us much to reconstruct the problem. A typical example is the so-called Subset Sum Problem: given a set of natural numbers \(s\), is there a subset \(s_i \subseteq s\) that adds up to \(k\)? Now consider the following statement:

**Statement 1** \(a\) is the \(n\)-th subset of \(S\) that adds up to \(k\).

Here "\(a\)" is the name of a set and "the \(n\)-th subset of \(S\) that adds up to \(k\)" a unique description. Note that we can compute the unique description effectively when we have the name and vice versa: i.e. there exists a computable bijection between the set of names and the set of unique descriptions. We have an algorithm to solve the search problem corresponding with the statement and the associated decision problem effectively:

1. Search problem: What is the \(n\)-th subset of \(S\) that adds up to \(k\)?
2. Decision problem: Does the \(n\)-th subset of \(S\) that adds up to \(k\) exist?

A central question is:

**Question 1.** Are there uniquely identifying descriptions of objects that contain more or less information than the names of the objects they denote?

The prima facie answer to this question is no. If we can compute the name from the description and vice versa, independent of the amount of time this takes, the descriptive complexities should stay close to each other in the limit.

---

1 For an elaborate analysis of this example, consult the last part of the Appendix in paragraph 8.
On the other hand, observe that computable bijections are by definition information efficient. When the information efficiency of a function is not well-defined, the bijection is also not well defined. We call this the Principle of Characteristic Information Efficiency: if two computations have a different information efficiency, then different functions are involved in their computation. As we have seen this is the case for addition. In this particular case there is no single finite mathematical function that describes the information efficiency of the bijection between sets and the sums of their subsets. There are infinitely many in the limit. The unique description “the first subset of \( s \) that adds up to \( k \)” is ad hoc. It has no clear relation with the name of the denoted subset \( a \). This kind of ad hoc unique descriptions are abundant in everyday life. Especially in relation to expressions like “the first \( x \) that ... ” or “the \( n \)-th \( x \) that ...”. Consider the descriptions:

1. The first four-leaf clover I’ll find this afternoon.
2. The 25-th man with a moustache I’ll see in the city.

It is clear the description “The first four-leaf clover I’ll find this afternoon” does not describe an intrinsic property of a certain plant. Actually, which plant I’ll find (if I find one) completely depends on my search method. The descriptive complexity, of the plant from my perspective at the moment I utter the phrase is a combination of the amount of information in the description itself and the description of the search method I’m going to use. Suppose I throw a dice to select my path when roaming around the city to find guys with moustaches, then the description the search process to identify “The 25-th man with a moustache I’ll see in the city” can easily contain more information than the descriptive complexity of his name.

According to this analysis the answer to research question 1 is positive: if a description of the search process, plus a partial description of the object, determines the object we will find, then this description possibly is more complex than the name of the object itself. At the same time the unique description itself can contain much less information. Mutatis mutandis, there is no way that we can search systematically for the “the first subset of \( s \) that adds up to \( k \)” using the information given in \( k \). A fortiori there is no search process that works in time polynomial to the complexity of the search problem. The search process creates the object I will find.

Of course, given the number \( k \) we can always find a subset that adds up to \( k \) by simply enumerating all possible subsets ordered by cardinality and compute the sum. In this case we are not using the specific information given in \( k \) in the organization of the search process. Such a search by enumeration simply generates the missing information about \( s_1 \) given \( s \) by a process of counting.

A question that emerges is whether there are sets for which there is no faster way to find a solution than by pure enumeration of the possible solutions. We prove this for a (prima facie) relatively simple countable object: the set of natural numbers \( \mathbb{N} \) and its finite subsets. We write this set as \( \mathcal{P}(\mathbb{N}) \). We investigate two mappings to a two dimensional infinite discrete space \( \mathbb{N}^2 \) (think of a chess board that extends to infinity on two sides):
We show (via an elaborate counting argument) that $\mathcal{P}(\mathbb{N})$ can be mapped efficiently onto $\mathbb{N}^2$. In every cell there is exactly one finite set of natural numbers and vice versa.

We observe that all possible descriptions of the form “the $n$-th subset of $\mathbb{N}$ that adds up to $k$” can be mapped trivially onto the plane $\mathbb{N}^2$: here $n$ is a column and $k$ is a row. In every cell there is exactly one unique description of a set. Not all unique descriptions will denote, e.g. “the 10-th set that adds up to 2” does not exist. The infinite space $\mathbb{N}^2$ is compressed infinitely over the $y$-axis.

Comparable mappings form the core of Cantor’s argument that proves the existence of superfine sets. Our argument follows a related strategy. The Cantor packing function maps the set of natural numbers $\mathbb{N}$ onto the discrete plane $\mathbb{N}^2$. Using this construction we can investigate all possible mappings between sets and their descriptions in terms of elastic translations over the two dimensional space. We show that all possible mappings between names of sets and their descriptions are unboundedly information expanding. The description of most typical sets $s$ as “the $n$-th subset of $\mathbb{N}$ that adds up to $k$” using the numbers $n$ and $k$ contains more information than the index of $s$ itself. Another way of formulating this insight is that the information in the natural numbers does not grow fast enough to characterize the set of descriptions. The set of descriptions is semi-countable: their complexity “outruns” the information in the set of natural numbers unboundedly in the limit. The set $\mathbb{N}$ is not rich enough to describe semi-countable sets. Consequently search by enumeration is the fastest way to construct them algorithmically.

1.2 Overview of the paper

We start with a conceptual overview of various types of computational processes: primitive recursive, $\mu$-recursive and non-deterministic. We show that $\mu$-recursive processes have a special status in so far that they allow for unbounded counting. We show that these processes generate information in logarithmic time in special circumstances.

We analyze this insight in the context of Kolmogorov complexity and Levin complexity and observe that these measures are not accurate enough for our purpose. We propose the concept of Information Efficiency of functions as an alternative complexity measurement theory. We give a detailed analysis of the recursive functions. We observe that information efficiency is not associative for addition.

We study the Cantor pairing function as an information preserving bijection between $\mathbb{N}$ and $\mathbb{N}^2$. We study the information efficiency of elastic translations over Cantor bijections over the space $\mathbb{N}^2$. We show that there is a spectrum of these translations.

We show that the set $\mathcal{P}(\mathbb{N})$, of all finite sets of natural numbers, can be mapped onto $\mathbb{N}^2$ efficiently. This allows us to investigate the general conditions for elastic translations based on addition and multiplication of sets of numbers.
The object that describes all possible additions of finite sets of natural numbers is \( \phi_\Sigma \). The corresponding object for multiplication is \( \phi_\Pi \). We show that the resulting set of unique descriptions of sets “the \( n \)-th subset of \( \mathbb{N} \) that adds up to \( k \)” is

1. associated with an infinite number of different computations.
2. not fully characterized by the set of natural numbers: the information in the descriptions grows faster than any counting process.

The set \( \phi_\Sigma \) is semi-countable. We can search sets in exponential time but not in polynomial time. The object \( \phi_\Pi \) is fundamentally less complex as a result of the fact that the information efficiency for multiplication is associative.

This argument can easily be generalized to the Subset Sum problem, which proves the separation between \( N \) and \( NP \).

2 Conceptual Analysis

In this paragraph we give a conceptual analysis of the issues concerning the interaction between information and computation. For a more global discussion of the underlying philosophical problems the reader is referred to [5]. We will use the prefix free Kolmogorov complexity \( K(x) \) as our measure of descriptive complexity of a string \( x \) and [3] as basic reference: \( K(x) \) is the length of the shortest program that computes \( x \) on a reference Universal Turing machine.

2.1 Types of Computational Processes

There are at least three fundamentally different types of computing (See Figure 1):

- **Elementary deterministic computing** as embodied in the primitive recursive functions. This kind of computing does not generate information: the amount of information in the Output is limited by the sum of the descriptive complexity of the Input and the Program.
- **Deterministic computing enriched with search** (bounded or unbounded) as embodied by the class of Turing equivalent systems, specifically the \( \mu \)-recursive functions. This type of computing generates information at logarithmic speed; the amount of information in the Output is not limited by the sum of the descriptive complexities of the Input and the Program.
- **Non-deterministic computing generates** information at linear speed.

Suppose there is a class of search problems with a polynomial time checking function that cannot be solved by a deterministic program but can be solved using bounded search. Such a search routine would take exponential time, since information generation has logarithmic speed. A non-deterministic computer could generate (guess) the required information at linear speed and then perform the test in polynomial time. The existence of such a class of search problems
would indicate a separation between $P$ and $NP$ for the associated decision problems: these problems cannot be solved deterministically, they can be solved using bounded search in exponential time and non-deterministically in polynomial time.

This analysis shows that the formulation of search problems in terms of polynomial time bounds and Turing Machines might be quite misleading. More important than the polynomial time bound is the fact that the search functions can not be computed at all (in general) by deterministic functions, while the distinction between deterministic search and primitive recursion is hard to make in the context of Turing machines. We analyse this issue in the following paragraph.

2.2 The $\mu$-operator for unbounded search

There is a subtle difference between systematic search and deterministic construction that is blurred in our current definitions of what computing is. If one considers the three fundamental equivalent theories of computation, Turing machines, $\lambda$-calculus and recursion theory, only the latter defines a clear distinction between construction and search, in terms of the difference between primitive recursive functions and $\mu$-recursive functions. The set of primitive recursive functions consists of: the constant function, the successor function, the projection function, composition and primitive recursion. With these we can define everyday mathematical functions like addition, subtraction, multiplication, division, exponentiation etc. In order to get full Turing equivalence one must add the
$\mu$-operator. In the world of Turing machines this device coincides with infinite loops associated with undefined variables. It is defined as follows in [4]:

For every 2-place function $f(x, y)$ one can define a new function, $g(x) = \mu y [f(x, y) = 0]$, where $g(x)$ returns the smallest number $y$ such that $f(x, y) = 0$. Defined in this way $\mu$ is a partial function. One way to think about $\mu$ as in terms of an operator that tries to compute in succession all the values $f(x, 0)$, $f(x, 1)$, $f(x, 2)$, ... until for some $m$, $f(x, m)$ returns 0, in which case such an $m$ is returned. In this interpretation, if $m$ is the first value for which $f(x, m) = 0$ and thus $g(x) = m$, the expression $\mu y [f(x, y) = 0]$ is associated with a routine that performs exactly $m$ successive test computations of the form $f(x, y)$ before finding $m$. Since the $\mu$-operator is unbounded $m$ can have any value.

Note that the name $g$ does not refer to a function but to a function-scheme. The $x$ in the expression $g(x)$ is not an argument of a function but the index of a function name $f_x(y) \iff f(x, y)$. We can interpret the $\mu$-operator as a meta-operator that has access to an infinite number of primitive recursive functions. In this interpretation there is no such thing as a general search routine. Each search function is specific: searching for your glasses is different from searching for your wallet, even when you look for them in the same places.

The difference between primitive recursion and $\mu$-recursion formally defines the difference between construction and search. Systematic search involves an enumeration of all the elements in the search space together with checking function that helps us to decide that we have found what we are looking for. We will have to look into this conception of enumeration in more depth.

![Fig. 2. Counting Processes](image-url)
2.3 Determinism versus non-determinism

In this paragraph we discuss the hybrid nature of unary counting processes, which, in a manner of speaking, are positioned between fully deterministic and non-deterministic processes. By definition deterministic processes do not generate new information, because the outcome of the process is determined. For a full discussion of this issue see [5]. We start with a detailed analysis of the seven elementary counting processes (A-G) shown in figure 2. The tensor operation \( \otimes \) signifies concatenation.

- Automaton A is deterministic and it does not halt. It starts with an empty string \( \epsilon \) and writes an infinite sequence of ones.
- Automaton B is non-deterministic. It generates the set of all finite strings of ones, i.e. the set of all finite unary numbers. We will call this a Counting Automaton or CA.
- Automaton C, also known as the Coin Flipping Automaton or CFA, is non-deterministic. It generates the set of all finite binary strings consisting of zeros and ones, i.e. the set of all finite binary numbers.
- Automaton D is deterministic. It is equivalent to automaton B with the addition of an extra test that checks the Kolmogorov complexity of the string \( x \) generated so far. As soon as \( x \) has a complexity greater than a constant \( c \) the process stops and produces output \( x \). We will ignore for the sake of argument that the Kolmogorov complexity is not computable and assume that there is some oracle that gives us a decision on the matter.
- Automaton E is non-deterministic. It is equivalent to automaton C with the addition of an extra test that checks the Kolmogorov complexity of the string \( x \) generated so far. As soon as \( x \) has a complexity greater than a constant \( c \) the process stops and produces output \( x \).
- Automaton F is deterministic. It is equivalent to automaton D, but now the test routine involves a computable function \( f(x, y) = 0 \) running on an input index \( y \). In fact it is an implementation of the central routine of the \( \mu \)-recursive search process that we discussed in the previous paragraph. It defines \( \mu \)-recursion based on a Counting Automaton.
- Automaton G is non-deterministic. It is equivalent to automaton E. Here also the test routine involves a computable function \( f(x, y) = 0 \) running on an input index \( y \). It defines \( \mu \)-recursion based on a Coin Flipping Automaton.

The difference between automaton A and B illustrates the fact that counting is essentially a non-deterministic operation. Automaton A does not effectively generate an object, whereas B generates all finite unary strings. Consequently the amount of information that B generates is unbounded. The information is generated by a sequence of free binary decisions to continue counting followed by one decision to stop the process.

The decisions to stop and start the process can be seen as meta-decisions that as such are not an intrinsic part of the process. This is illustrated by the fact that as soon as we add a stop criterion in automaton D and E the unary
counting process becomes deterministic and the binary string generation keeps its non-deterministic nature.

The fact that the systems \( B \) and \( C \) both generate information is illustrated by systems \( D \) and \( E \). Both stop at the moment when a certain amount of information of size \( c \) is generated. Since unary strings code information very inefficiently, and thus have a low Kolmogorov complexity, process \( D \) needs to perform at least \( 2^c \) write operations before it stops, where process \( E \) can reach this goal in principle in \( c \) steps. In this case the computation \( D \) has exponential time whereas \( E \) can work in linear time. The observations we can make on the basis of this analysis are:

**Observation 1**  — Only non-deterministic processes generate information. Deterministic processes by definition do not generate information \([1]\).

– Both unary counting and coin flipping are non-deterministic processes.

– Counting generates information at logarithmic speed. The time needed to generate a certain amount of information by means of unary counting, is exponential in the amount of time a coin flipping automaton needs to generate the same amount of information.

Unary counting with a stop criterion is hybrid in the sense that it has characteristics of both deterministic and non-deterministic processes. This explains the special status of these kind of processes in recursion theory. Unary counting with a stop criterion is a form of computing that is essentially stronger than standard deterministic computing. There are many conceptual problems around this notion of computing, one of which is the fact that the descriptive complexity of the computational process at any time during the computation may be much higher than the complexity of the actual output. See \([1]\) for a discussion. The crucial limiting factor is the descriptive complexity of the halting test: \( K(x) < c \) for processes \( D \) and \( E \) and \( f(x, y) = 0 \) for processes \( F \) and \( G \). A central observation in this context is the so-called non-monotonicity of set theoretical operations (see \([5]\), par. 6.2).

Since \( \mu \)-recursion is stronger than primitive recursion there will be classes of search problems that can be solved by \( \mu \)-recursive functions and not in general by primitive recursive functions. The search process in \( \mu \)-recursion is driven by counting. Consequently a non-deterministic version of \( \mu \)-recursion using a Coin Flipping Automaton could solve a search problem, i.e. compute the value \( m \) for the test function \( g(m) \), in time linear in the length of representation of the number \( m \), while the search process in the classical deterministic \( \mu \)-recursion, using a Counting Automaton, would take time exponential in the length of the representation of \( m \), i.e. the value of \( m \). Moreover if the time complexity of the computation \( f(x, y) \) is polynomial in the length of the input then the fact that a solution can be generated non-deterministically in linear time would be overshadowed by the time complexity of the checking function.
3 Information Efficiency of Recursive Functions

Let $x, y, z \in \mathbb{N}$, where $\mathbb{N}$ denotes the natural numbers and we identify $\mathbb{N}$ and \{0,1\}$^*$ according to the correspondence

$$(0, \varepsilon), (1, 0), (2, 1), (3, 00), (4, 01), \ldots$$

Here $\varepsilon$ denotes the empty word. The length $l(x)$ of $x$ is the number of bits in the binary string $x$. In the following we will use the logarithm with base 2 as our standard reference $\log x = \log_2 x$. The standard reference \[3\] for the definitions concerning Kolmogorov complexity is followed. $K$ is the prefix-free Kolmogorov complexity of a binary string. It is defined as:

**Definition 1.**

$$K(x|y) = \min_i \{l(i) : U(iy) = x\}$$

i.e. the shortest self-delimiting index of a Turing machine $T_i$ that produces $x$ on input $y$, where $i \in \{1, 2, \ldots\}$ and $y \in \{0,1\}^*$. Here $l(i)$ is the length of a self-delimiting code of an index and $U$ is a universal Turing machine that runs program $y$ after interpreting $i$. The length of $l(i)$ is limited for practical purposes by $n + 2 \log n + 1$, where $n = |i|$. The actual Kolmogorov complexity of a string is defined as the one-part code that is the result of, what one could call, the:

**Definition 2 (Forcing operation).**

$$K(x) = K(x|\varepsilon)$$

According to the classical view the descriptive complexity of the Output of a deterministic computational process is bounded by the sum of the complexity of the Input and the Program (See Figure [1]):

$$K(Output) \leq K(Input) + K(Program) + O(1) \tag{1}$$

Based on the discussion above we formulate the conjecture:

**Conjecture 1.** Deterministic computational processes generate information at logarithmic speed.

Acceptance of this conjecture would imply a shift from Kolmogorov complexity to Levin complexity which takes the influence of the computing time into account:

**Definition 3 (Levin complexity: Time = Information).** The Levin complexity of a string $x$ is the sum of the length $l(p)$ and the logarithm of the computation time of the smallest program $p$ that produces $x$ when it runs on a universal Turing machine $U$, noted as $U(p) = x$:

$$K_t(x) = \min_p \{l(p) + \log(time(p)), U(p) = x\}$$
The inequality then becomes:

\[ K_i(Output) \leq K(Input) + K(Program) + \log(time(Program)) + O(1) \] (2)

The problem with such a proposal is that our classical proof techniques and information measures are not sensitive enough to observe the difference between the two measures in practical situations. Information production at logarithmic speed is extremely slow and we will in every day life never sense the way it influences our measurements. The situation is not unlike the one in the theory of relativity. Our measurement of time is affected by our relative speed, but in every day life the speeds at which we travel in relation to the accuracy of our measurement techniques are such that we do not observe these fluctuations. The same holds for the difference between Kolmogorov and Levin complexity. First of all both measures are uncomputable, so we can never present a convincing example illustrating the difference. Secondly, if we estimate the number of computational steps the universe has made since the Big Bang as \(10^{123}\), then a deterministic system would have produced about 400 bits of information in this time span. Even if we could compute the value of \(K(Output)\) and \(K_i(Output)\) then the asymptotic nature of the measure, reflected in the \(O(1)\) parameter, which is related to the descriptive complexity of the reference Universal machines we are considering, does not allow us to reach the accuracy necessary to observe the difference on the time scale of our universe as a whole. So the classical theory of Kolmogorov complexity is of little use to us. We need to develop more advanced information measurement techniques.

One possibility is to shift our attention from Turing machines to recursive functions. The big advantage of recursive functions is that we have a reliable definition of primitive recursive functions, which relieves us of the burden to select something like a reference Universal machine, which eliminates the asymptotic nature of the measurement theory we get. Secondly we can model the flow of information through computational processes more effectively. We define:

**Definition 4 (Information in Natural numbers).**

\[ \forall (x \in \mathbb{N}) I(x) = \log x \]

The rationale behind the choice of the log function as an information measure is discussed extensively in [5]. The big advantage of the definition of an information measure using recursive functions is the fact that we can get rid of the asymptotic \(O(1)\) factor, since we do not have to relativize over the class of universal machines. We get a theory about compressible numbers, much in line with Kolmogorov complexity, if we introduce the notion of the information efficiency of a function. The **Information Efficiency** of a function is the difference between the amount of information in the input of a function and the amount of information in the output. We use the shorthand \(f(x)\) for \(f(x_1, x_2, \ldots, x_k)\). We consider functions on the natural numbers. If we measure the amount of information in a number \(n\) as:

\[ I(n) = \log n \]
then we can measure the information effect of applying function $f$ to $n$ as:

$$I(f(n)) = \log f(n)$$

This allows us to estimate the information efficiency as:

$$\delta(f(n)) = I(f(n)) - I(n)$$

More formally:

**Definition 5 (Information Efficiency of a Function).** Let $f : \mathbb{N}^k \to \mathbb{N}$ be a function of $k$ variables. We have:

- the input information $I(\pi)$ and
- the output information $I(f(\pi))$.
- The information efficiency of the expression $f(\pi)$ is

$$\delta(f(\pi)) = I(f(\pi)) - I(\pi)$$

- A function $f$ is information conserving if $\delta(f(\pi)) = 0$ i.e. it contains exactly the amount of information in its input parameters,
- it is information discarding if $\delta(f(\pi)) < 0$ and
- it has constant information if $\delta(f(\pi)) = 0$.
- it is information expanding if $\delta(f(\pi)) > 0$.

The big advantage of this definition over Kolmogorov complexity is that we can compute the flow of information through functions exactly. In the Appendix I in paragraph 8 we give extensive examples of the information efficiency of elementary recursive functions.

**Definition 6 (Principle of Characteristic Information Efficiency).** The concept of information efficiency is characteristic for a function. Consequently if the information efficiency varies over sets of computable numbers, then different functions must be involved in their computation.

The concept of Information Efficiency gives us a tool to decide between Kolmogorov complexity and Levin complexity as the right measures for information. We make the following observation:

**Lemma 1.** If we can define a computable bijection $p : \mathbb{N} \to \mathbb{N}$ for which in the limit the information efficiency is unbounded then computational processes generate information beyond the information stored in the program $p$ itself, i.e. equation 1 is invalid. Consequently equation 2 is the right bound.

Proof: First observe that most natural numbers are typical, i.e. random. Their Kolmogorov complexity is “close” to the logarithm of the value. Since $p$ computes a bijection on $\mathbb{N}$, both input and output will contain a “sufficient” amount of random, i.e. incompressible, elements to the effect that the equations 1 and 2 describe equalities for dense sets at any scale. Now consider equation 1 which can be rewritten as:
\[ K(\text{Output}) - K(\text{Input}) = \]
\[ \delta(\text{Program}(\text{Input})) \leq K(\text{Program}) + O(1) = c \quad (3) \]
i.e. the information efficiency is bounded which contradicts the assumption.

Now rewrite (2) as:

\[ K_t(\text{Output}) - K(\text{Input}) = \]
\[ \delta(\text{Program}(\text{Input})) \leq K(\text{Program}) + \log(\text{time}(\text{Program})) + O(1) \quad (4) \]

with the extra factor \( \log(\text{time}(\text{Program})) \). This is the right bound since, following observation 1, \( \log(\text{time}(\text{Program})) \) is the maximum speed at which deterministic computational processes generate information. □

4 Computable functions that generate and discard information

In this paragraph we show that there are indeed finite halting programs that generate and discard an unbounded amount of information in the limit. Central is the notion of an elastic transformation of the set \( \mathbb{N}^2 \).

4.1 The Cantor packing function

Observe that there is a two-way polynomial time computable bijection \( \pi : \mathbb{N}^2 \to \mathbb{N} \) in the form of the so-called Cantor packing function:

\[ \begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
0 & 0 & 2 & 5 & 9 & 14 & 20 & 27 & 35 & 44 & 54 \\
1 & 1 & 4 & 8 & 13 & 19 & 26 & 34 & 43 & 53 & 69 \\
2 & 3 & 7 & 12 & 18 & 25 & 33 & 42 & 52 & 63 & 75 \\
3 & 6 & 11 & 16 & 21 & 26 & 32 & 39 & 46 & 53 & 69 \\
4 & 10 & 16 & 23 & 31 & 40 & 50 & 61 & 73 & 86 & 100 \\
5 & 15 & 22 & 30 & 39 & 49 & 60 & 72 & 85 & 99 & 114 \\
6 & 21 & 29 & 38 & 47 & 58 & 70 & 83 & 97 & 112 & 128 \\
7 & 28 & 37 & 47 & 58 & 70 & 83 & 97 & 112 & 128 & 145 \\
8 & 36 & 46 & 57 & 69 & 82 & 96 & 111 & 127 & 144 & 162 \\
9 & 45 & 56 & 68 & 81 & 95 & 110 & 126 & 143 & 161 & 180 \\
\end{array} \]

\textbf{Fig. 3.} An initial segment of the Cantor packing function
\[
\pi(x, y) := \frac{1}{2} (x + y)(x + y + 1) + y
\]  

(5)

An example of the computations involved in the bijection is given in the Appendix in paragraph [9]. The Fueter - Pólya theorem [2] states that the Cantor pairing function and its symmetric counterpart \( \pi'(x, y) = \pi(y, x) \) are the only possible quadratic pairing functions. A segment of this function is shown in figure [2]. The information efficiency of this function is:

\[
\delta(\pi(x, y)) = \log\left(\frac{1}{2} (x + y + 1)(x + y) + y\right) - \log x - \log y
\]  

(6)

![Fig. 4](image)

**Fig. 4.** The information efficiency of the Cantor packing function, \(0 < x < 10^9\), \(0 < y < 10^9\), \(-1 < z < 7\). The shaded area is the \(z = 0\) surface.

**Observation 2** The Cantor function defines what one could call: a discontinuous folding operation over the counter diagonals. On the line \(y = 0\) we find the images \(1/2x(x - 1) = \Sigma^c\). For points \((x_1, y_1)\) and \((x_2, y_2)\) on different counter diagonals \(x_1 + y_1 = c_1\) and \(x_2 + y_2 = c_2\) we have that \(|\pi(x_1, y) - \pi(x_2, y)| = c\). Equation [6] can be seen as the description of an information topology. The Cantor function runs over the counter diagonals and the image shows that the information efficiencies of points that are in the same neighborhood are also close.

**Observation 3** The information efficiency of the Cantor packing function has infinite precision (see Figure [4]).
This is what one would expect from a function that defines a polynomial time computable bijection. We analyse some limits that define the information efficiency of the function. On the line $y = x$ we get:

$$\lim_{x \to \infty} \delta \pi(x, x) = \lim_{x \to \infty} \log \left( \frac{1}{2}(2x + 1)(2x) \right) - 2 \log x = \frac{2x^2 + x}{x^2} = 1$$

For the majority of the points in the space $\mathbb{N}^2$ the function $\pi$ has an information efficiency close to one bit. On every line through the origin $y = hx$ ($h > 0$) the information efficiency in the limit is constant:

$$\lim_{x \to \infty} \delta(\pi(x, hx)) = \log(1/2(h + 1)^2) - \log h$$

Yet on every line $y = c$ (and by symmetry $x = c$) the information efficiency is unbounded:

$$\lim_{x \to \infty} \delta(\pi(x, c)) = \infty$$

Together the equations 5, 6, 7, 8 and 9 characterize the basic behavior of the information efficiency of the Cantor function.

### 4.2 Linear elastic transformations of the Cantor space

Actually equation 9 is responsible for remarkable behavior of the Cantor function under what one could call elastic transformations. For elastic transformations we can compress the Cantor space along the $y$-axis by any constant without actually losing information. Visually one can inspect this counter intuitive phenomenon in figure 4 by observing the concave shape of the information efficiency function: at the edges ($x = 0, y = 0$) it has in the limit an unbounded amount of compressible information. The source of this compressibility in the set $\mathbb{N}$ is the set of numbers that is logarithmically close to sets $\frac{1}{2}x(x + 1)$ and $\frac{1}{2}y(y + 1) + y$). In terms of Kolmogorov complexity these sets of points define regular dips of depth $\frac{1}{2} \log x$ in the integer complexity function that in the limit provide an infinite source of highly compressible numbers. In fact, when we would draw figure 4 at any scale over all functions $f : \mathbb{N}^2 \to \mathbb{N}$ we would see a surface with all kinds of regular and irregular elevations related to the integer complexity function.

Observe figure 5. The upper part shows a discrete translation over the $x$-axis by a factor 2. This is an information expanding operation: we add the factor 2,
Fig. 5. Row expansion by factor 2 and elastic translation by a factor 2 for the segment in Figure 3. For a computation of the exact information efficiency see figure 6.

![Linear column expansion by factor 2](image_url)

![Linear elastic translation by factor 2: Linear column expansion + row compression](image_url)

Fig. 6. An exact computation of the point by point information efficiency after an elastic transformation by a factor 2 for a segment (1-4 by 1-19) of the table in figure 5. The existence of two separate interleaving information efficiency functions is clearly visible in line with definition 6.

i.e. one bit of information, to each $x$ coordinate. Since we expand information, the density of the resulting set in $\mathbb{N}^2$ also changes by a factor 2. In the lower
Fig. 7. A fragment of the bijection generated by an elastic transformation by a factor 2 for a segment (0-4 by 0-9) of the table in figure 5. The bijection $\pi(\epsilon_2(\pi^{-1})) : N \rightarrow N$ generates a cloud of compressible and expandable points. The compressible points are above the line $x = y$.

Part we have distributed the values in the columns 0, 2, 4, … over the columns $0 - 1, 2 - 3, 4 - 5, \ldots$. We call this an elastic translation by a factor 2. The space $10 \times 10 = 100$ is transformed into a $5 \times 20 = 100$ space. The exact form of the translation is: $\epsilon_2(x, y) = (2x + (y \text{ mod } 2), \lfloor \frac{y}{2} \rfloor)$.

The effect of this elastic translation on the information efficiency on a local scale can be seen in figure 6. After some erratic behavior close to the origin the effect of the translation evens out. There are traces of a phase transition: close to the origin the size of the $x, y$ coordinates is comparable to the size of the shift $c$, which influences the information efficiency considerably. From the wave pattern in the image it is clear that a linear elastic transformation by a factor $c$ essentially behaves like a set of $c$ functions (in this case 2), each with a markedly different information efficiency.

Even more interesting is the behavior, shown in figure 7, of the bijection:

$$N \xrightarrow{\pi^{-1}} N^2 \xrightarrow{\epsilon_2} N^2 \xrightarrow{\pi} N$$

Although the functions $\pi$, $\pi^{-1}$ and $\epsilon_2$ are bijections and can be computed point wise in polynomial time, all correlations between the sets of numbers seem to have been lost. The reverse part of the bijection shown in formula 11 seems hard to compute, without computing large parts of 10 first.

$$N \xrightarrow{\pi^{-1}} N^2 \xrightarrow{\epsilon_2^{-1}} N^2 \xrightarrow{\pi} N$$

(11)
Fig. 8. The information efficiency of the reference function of the Cantor packing function on the same area as in figure 4 after an elastic shift by a factor $100$, $0 < x < 10^9$, $0 < y < 10^9$, $-1 < z < 10.6$. The shaded area is the $z = 0$ surface.

Observation 4 Linear elastic transformations introduce a second type of horizontal discontinuous folding operations over the columns. These operations locally distort the smooth topology of the Cantor function into clouds of isolated points.

On a larger scale visible in figure 8 we get a smooth surface. The distortion of the symmetry compared to figure 4 is clearly visible. In accordance with equation 9 nowhere in the set $\mathbb{N}^2$ the information efficiency is negative. In fact the information efficiency is lifted over almost the whole surface. On the line $x = y$ the value in the limit is:

$$\lim_{x \to \infty} \delta \pi(\epsilon_2((x, x))) =$$

$$\lim_{x \to \infty} \log\left(\frac{1}{2}(2x + x/2 + 1)(2x + x/2 + x/2) - \log 2x - \log x/2\right) = 2 \log 5 - 3$$

A more extreme form of such a distortion can be seen in figure 9 that shows the effect on the information efficiency after an elastic translation by factor 100. Clearly the lift in information efficiency over the whole surface can be seen. We only show the first of 100 different information efficiency functions here. Computed on a point by point basis we would see periodic saw-tooth fluctuations over the $x$-axis with a period of 100. This discussion shows that elastic transformations of the Cantor space act as a kind of perpetuum mobile.
of information creation. For every elastic transformation by a constant \( c \) the information efficiency in the limit is still positive:

**Lemma 2.** No compression by a constant factor \( c \) along the \( y \)-axis (or \( x \)-axis, by symmetry) will generate a negative information efficiency in the limit.

Proof: immediate consequence of equation \([9]\). The information efficiency is unbounded in the limit on every line \( x = c \) or \( y = c \). \( \square \)

### 4.3 A general model of elastic transformations

In the following we will study, what we call *general elastic transformations* of the space \( \mathbb{N}^2 \):

**Definition 7.** The function \( \epsilon_f : \mathbb{N}^2 \to \mathbb{N}^2 \) defines an elastic translation by a function \( f \) of the form:

\[
\epsilon_r(x, y) = (xf + (y \mod r(x)), \lfloor \frac{y}{r(x)} \rfloor)
\]

Such a transformation is super-elastic when:

\[
\lim_{x \to \infty} r(x) = \infty
\]
It is polynomial when it preserves information about \( x \):

\[
\lim_{x \to \infty} r(x) = cx^k
\]

It is linear when:

\[
r(x) = c
\]

The reference function of the translation is:

\[
\epsilon_r'(x, y) = (r(x)x, \frac{y}{r(x)})
\]

We will assume that the function \( f \) can be computed in time polynomial to the length of the input.

Observe that the reference function: \( \epsilon_r'(x, y) = (r(x)x, \frac{y}{r(x)}) \) is information neutral on the arguments:

\[
\log x + \log y - (\log r(x)x + \log \frac{y}{r(x)}) = 0
\]

An elastic translation consists from an algorithmic point of view of two additional operations:

1. An **information discarding** operation on \( y \):

\[
\frac{y}{r(x)} \rightarrow \lfloor \frac{y}{r(x)} \rfloor.
\]

2. An **information generating** operation on \( x \):

\[
r(x)x \rightarrow r(x)x + (y \mod r(x)).
\]

A schematic overview of a linear elastic shift is given in figure 10. Here the letters \( a, \ldots i \) are natural numbers. An arbitrary point in neighborhood \( \alpha \) with coordinates \((b, f)\) close to the diagonal is translated to point \((a, h)\) in neighborhood \( \beta \). The formula for the translation is given by definition 7:

\[
\epsilon_r(x, y) = (r(x)x + (y \mod r(x)), \lfloor \frac{y}{r(x)} \rfloor).
\]

We have \( a = \lfloor \frac{b}{r(f)} \rfloor \) and \( h = r(f)e + (b \mod r(f)) \).

The information efficiency of an elastic transformation is:

\[
\delta(\pi(\epsilon_r)(x, y)) = \delta(\pi(r(x)x + (y \mod r(x)), \lfloor \frac{y}{r(x)} \rfloor))
\]

The information efficiency of a linear elastic transformation by a factor \( f(x) = c \) is:

\[
\delta(\pi(cx + (y \mod r(x)), \lfloor \frac{y}{c} \rfloor))
\]

**Observation 5** Elastic transformations by a constant \( c \) of the Cantor space replace the highly efficient Cantor packing function with \( c \) different interleaving functions, each with a different information efficiency. Equation 15 must be seen as a meta-function or meta-program that spawns off \( c \) different new programs.

This is illustrated by the following lemma:

**Lemma 3.** Linear elastic translations generate information.
Fig. 10. An elastic translation from neighborhood $\alpha$ to neighborhood $\beta$.

Proof: This is an immediate effect of the use of the mod function. A linear elastic translation by a constant $c$ of the form:

$$\epsilon_c(x,y) = (cx + (y \mod c), \lfloor \frac{y}{c} \rfloor)$$

has $c$ different information efficiency functions. Note that the function $x \mod c$ produces all numbers $d < c$, including the incompressible ones that have no mutual information with $c$: $K(d|c) = \log d + O(1)$. For each value $d = y \mod c$ we get a function with different information efficiency:

$$\delta(\pi(\epsilon_{c,d}(x,y))) = \log \pi(cx + d, \lfloor \frac{y}{c} \rfloor) - \log x - \log y$$

On a point by point basis the number $d$ is part of the information computed by $\pi(\epsilon_{c,d}(x,y)))$. In other words the computation $\pi(\epsilon_{c,d}(x,y))$ adds information to the input for specific pairs $(x,y)$ that is not available in the formula for $\epsilon_c(x,y)$. The effect is for linear transformations constant in the limit, so it is below the accuracy of Kolmogorov complexity. □

For typical cells $(x,hx)$ on a line $y = hx$ the function $\delta(\pi(\epsilon'_c(x,y)))$ gives in the limit a constant shift which can be computed as:

$$\lim_{x \to \infty} \frac{\pi(\epsilon'_c(x,y))}{\pi(x,y)} = \lim_{x \to \infty} \frac{1}{2} \frac{(cx + hx + 1)(cx + \frac{hx}{c}) + \frac{hx}{c}}{(cx + hx + 1)(x + hx) + hx} = \frac{(c^2 + h)^2}{c^2(1+h)^2} \geq 1 \quad (14)$$
Note that this value is only dependent on \( h \) and \( c \) for all \( c \in \mathbb{N}^+ \) and \( r \in \mathbb{R}^+ \). The general lift of the line \( x = y \) for an elastic shift by a constant \( c \) is:

\[
\lim_{x \to \infty} \log(\frac{1}{2} (cx + \frac{x}{c} + 1) (cx + \frac{x}{c} + \frac{x}{c})) - \log cx - \log \frac{x}{c} = (15)
\]

\[
-\log \frac{2}{c} - \log c + \log(2 + \frac{1}{c^2} + c^2)
\]

We get a better understanding of the extreme behavior of the reference function \( \delta(\pi(c'_c(x, y))) \) when we rewrite equation 14 as:

\[
\frac{(c^2 + h)^2}{c^2(1 + h)^2} = \frac{c^2}{1 + 2h + h^2} + \frac{2h}{1 + 2h + h^2} + \frac{h^2}{c^2 + 2c^2h + c^4h^2} \geq 1
\]

and take the following limit:

\[
\lim_{h \to \infty} \frac{(c^2 + h)^2}{c^2(1 + h)^2} = c^{-2}
\]

If \( c \) is constant then it has small effects on large \( h \) in the limit. The reference function allows us to study the dynamics of well-behaved “guide points” independent of the local distortions generated by the information compression and expansion operations. Note that elastic transformations start to generate unbounded amounts of information in each direction \( y = hx \) in the limit on the basis of equation 14:

\[
\lim_{c \to \infty} \frac{(c^2 + h)^2}{c^2(1 + h)^2} = \infty
\]

If \( c \) grows unboundedly then the information efficiency of the corresponding reference functions goes to infinity for every value of \( h \). Consequently, when \( c \) goes to infinity, the reference functions predict infinite information efficiency in \( \mathbb{N}^2 \) in all directions, i.e. we get infinite expansion of information in all regions without the existence of regions with information compression. This clearly contradicts central results of Komogorov complexity if we assume that elastic translations are defined in terms of a single program. The situation is clarified by the proof of lemma 3 if \( c \) goes to infinity we create an unbounded amount of new functions that generate an unbounded amount of information.

### 4.4 Polynomial transformations

The picture that emerges from the previous paragraphs is the following: we can define bijections on the set of natural numbers that generate information for almost all numbers. The mechanism involves the manipulation of clouds of points of the set \( \mathbb{N} \): sets with density close to the origin are projected into sparse sets of points further removed from the origin. This process can continue indefinitely.

In this context we analyse polynomial translations. The simplest example is the elastic translation by the factor \( x \):
Fig. 11. Row expansion by a superlinear, \( r(x) = x \) factor and a fragment of the corresponding elastic translation for the segment in Figure 3.

\[
\varepsilon_x(x, y) = (x^2 + (y \mod x), \left\lfloor \frac{y}{x} \right\rfloor)
\] (17)

A tiny fragment of the effects is shown in figure 11. There does not seem to be a fundamental difference compared to the previous examples and there seems to be no difficulty in constructing such a translation along the lines suggested in the figure. However, upon closer inspection things are different as can be observed in figure 12. The first table shows the computation of the function up to the number 32. Observe that the columns 14 and 15 are empty, while column 16–18 have a value. This effect does not appear in the second table. Linear elastic translations induce a change in the direction of the iso-information line \((d, g)\), but the image of the translation keeps a coherent topology at any stage of the computation.

Polynomial translations on the other side are discontinuous. They tear the space apart in to separate regions. An appropriate metaphor would be the following: expansion away from the origin over the \(x\)-axis, sucks a vacuum that must be filled by a contraction over the \(y\)-axis. Actually the creation of such a vacuum is an information discarding operation. The analysis above shows that the vacuum created by the shift described by equation 17 for cells on the line \(y = 0\) is bigger than the whole surface of the triangle \((o, d, g)\). The effect is that the image of the translation becomes discontinuous.

Theorem 1. Polynomial elastic transformations of \(\mathbb{N}^2\):
Fig. 12. Above: Super linear expansion by a function \( r(x) = x \) up to 32. Below: Linear elastic expansion by a factor 2 up to the number 32.

1. **Discard and expand information on the line \( y = 0 \) unboundedly.**
2. **Project a dense part of \( \mathbb{N}^2 \) and \( \mathbb{N} \) on \( y = 0 \).**
3. **Generate an unbounded amount of information for typical points in \( \mathbb{N}^2 \) in the limit.**

Proof: The formula for a polynomial translation is:

\[
\epsilon_r(x, y) = (cx^{k+1} + (y \mod cx), \left\lfloor \frac{y}{cx} \right\rfloor)
\]

Take \( k = c = 1 \). Consider a typical point \( f \) in figure 10. We may assume that the numbers \( d \) and \( g \) are typical (i.e. incompressible and thus \( \pi(g, d) \) is incompressible too. Remember that the Cantor function runs over the counter diagonal which makes the line \( (d, g) \) an iso-information line.

1. **Discard and expand information on the line \( y = 0 \) unboundedly:**
   - **Discard information:** Horizontally point \((f, 0)\) will be shifted to location \((f^2, 0)\) and \((f(+1), 0)\) to \((f^2 + 2f + 1, 0)\). The cantor index for point \((f, 0)\) is \( \frac{1}{2} f(f - 1) < f^2 \).
   - **Expand information:** The cells from \((f, 0)\) to \((f, 2f)\) will be “padded” in the strip between \((f^2, 0)\) and \((f^2 + 2f + 1, 0)\). But this operation “steals” a number of \( f \) cells from the domain above the line \((d, g)\). Now take a typical point \( m \) on location \((f, m)\), such that \( f < m < 2f \). This point will land at \((f^2 + m, 0)\) somewhere between \((f^2, 0)\) and \((f^2 + 2f + 1, 0)\),
which gives:

$$\pi(f,(f+m)) = \frac{1}{2}(f + (f + m) + 1)(f + (f + m)) + (f + m) =$$

$$2f^2 + \frac{1}{2}fm + m^2 + 3f + \frac{1}{2}(m + fm)$$

$$\gg f^2 + 2f + 1$$

Note that the effects are dependent on $f$ and $m$, so they are unbounded in the limit. Alternatively observe the fact that polynomial transformations are superelastic and apply equation 16.

2. Project a dense part of $\mathbb{N}^2$ and $\mathbb{N}$ on $y = 0$: For every point $f$ all the cells $(f,0)$ to $f(f,2f)$ up to the line $y = 2x$ will end up on line $y = 0$. since all points in this dense region are projected on the line $y = 0$ most points $\pi(f,(f+m))$ are incompressible.

3. Generate an unbounded amount of information for typical points in $\mathbb{N}^2$ in the limit. Take a typical point $(x,y)$ such that $\log x \approx \log y$:

$$\pi(\epsilon_r(x,y)) > \pi(\epsilon_r(x,1)) = \frac{1}{2}(x^2 + 2)(x^2 + 1) + 1 > \frac{1}{2}x^4$$

$$(18)$$

$$\delta(\pi(\epsilon_r(x,y))) > \log \frac{1}{2}x^4 - \log x - \log y \approx 4\log x - 2\log x = 2\log x$$

which gives $K\pi(\epsilon_r(x,y)) > K(\pi(x,y)) + 2\log x$.

This argument can easily be generalized to other values of $c$ and $k$. □

Polynomial shifts generate information above the asymptotic sensitivity level of Kolmogorov complexity. Note that $\epsilon_r(x,y)$ is still a computable bijection:

$$\forall (x,y) \in \mathbb{N}^2 \exists (u,v) \in \mathbb{N}^2 (\epsilon_r(u,v) = (x,y))$$

$$u^2 \leq x < (u + 1)^2$$

$$a = (u + 1)^2 - u^2$$

$$y = \left\lfloor \frac{v}{a} \right\rfloor$$

This analysis holds for all values $\delta(\pi(\epsilon_r(x,y))) \in \mathbb{N}$ including the values that are typical, i.e. incompressible.

**Observation 6** An immediate consequence is the translation $\epsilon_r(x,y) = (cx^{k+1} + (y \mod cx), \left\lfloor \frac{y}{a} \right\rfloor)$ must be interpreted as a function scheme, that produces a countable set of new functions. One for each column $x = c$. Actually $x$ can be seen as an index of the function that is used to compute $\epsilon_r(u,v) = (x,y)$. 
The difference between linear and polynomial elastic transformations marks the phase transition between continuous and non-continuous shifts. When we use the enumeration of the Cantor packing function to compute such shifts the time needed to compute the location of certain points on the line \( y = 0 \) close to each other may vary exponentially in the representation of the numbers involved. On the other hand such translations can be easily computed as bijections on a point by point basis in polynomial time, since the \( x \) coordinate of the image codes the index of the function that was used to compute it.

4.5 The information efficiency of arithmetical functions on sets of numbers

In this paragraph we study elastic translations that grow faster than any polynomial function based on elementary arithmetical functions. These translations are so “aggressive” that they destroy the topology of the space completely and project chaotic clouds of points on the \( x \)-axis. A crucial tool for the construction of super polynomial transformations is the bijective mapping of finite sets of numbers in \( \mathbb{N} \) to the space \( \mathbb{N}^2 \):

**Definition 8.** \( \mathcal{P}(\mathbb{N}) \) is the powerset or set of subsets of \( \mathbb{N} \). \( \mathcal{P}(\mathbb{N}) \) the set of finite subsets of \( \mathbb{N} \). A characteristic function of an infinite subset of \( S \subseteq \mathbb{N} \) is \( f_S : \mathbb{N} \to \mathbb{N} \), a monotone ascending function such that \( \forall (x)f_S(x+1) > f(x) \). Here \( x \) is the index of \( f_S(x) \) in \( S \).

\( \mathcal{P}(\mathbb{N}) \) is uncountable, whereas \( \mathcal{P}(\mathbb{N}) \) can be counted. Consequently \( \mathcal{P}(\mathbb{N}) - \mathcal{P}(\mathbb{N}) \) is also uncountable. Proofs of the countability of \( \mathcal{P}(\mathbb{N}) \) rely on the axiom of choice to distribute set \( \mathcal{P}(\mathbb{N}) \) into partitions with the same cardinality. A useful concept in this context is the notion of combinatorial number systems:

**Definition 9.** The function \( \sigma_k : \mathbb{N}^k \to \mathbb{N} \) defines for each element
\[
s = (s_k, \ldots, s_2, s_1) \in \mathbb{N}^k
\]
with the strict ordering \( s_k > \ldots s_2 > s_1 \geq 0 \) its index in a \( k \)-dimensional combinatorial number system as:
\[
\sigma_k(s) = \binom{s_k}{k} + \cdots + \binom{s_2}{2} + \binom{s_1}{1} \quad (19)
\]

The function \( \sigma_k \) defines for each set \( s \) its index in the lexicographic ordering of all sets of numbers with the same cardinality \( k \). The correspondence does not depend on the size \( n \) of the set that the \( k \)-combinations are taken from, so it can be interpreted as a map from \( \mathbb{N} \) to the \( k \)-combinations taken from \( \mathbb{N} \). For singleton sets we have: \( \sigma_1(x) = \binom{x}{1} = x, x \geq 0 \). For sets with cardinality 2 we have:
\[
\sigma_k(0) = \binom{1}{2} + \binom{0}{1} \to \{1, 0\}
\]
\[ \sigma_k(1) = \binom{2}{2} + \binom{0}{1} \to \{2, 0\} \]
\[ \sigma_k(2) = \binom{2}{2} + \binom{1}{1} \to \{2, 1\} \]
\[ \sigma_k(3) = \binom{3}{2} + \binom{0}{1} \to \{3, 0\} \ldots \]

We can use the notion of combinatorial number systems to prove the following result:

**Theorem 2.** There is a bijection \( \phi : \mathcal{P}(\mathbb{N}) \to \mathbb{N} \) that can be computed efficiently.

Proof: We prove the lemma for the set \( \mathbb{N}^+ \). Let \( S_k \) be the subset of all elements \( s \in \mathcal{P}(\mathbb{N}^+) \) with cardinality \( |s| = k \). For each \( k \in \mathbb{N} \) by definition 9 the set \( S_k \) is described by a combinatorial number system of degree \( k \). The function \( \sigma_k : \mathbb{N}^{+k} \to \mathbb{N} \) defines for each element \( s = (s_k, \ldots, s_2, s_1) \in \mathbb{N}^{+k} \) with the strict ordering \( s_k > \ldots > s_2 > s_1 \geq 0 \) its index in a \( k \)-dimensional combinatorial number system. By definition 9 the correspondence is a polynomial time computable bijection. Now define \( \phi^+_{|s|} : \mathcal{P}(\mathbb{N}) \to \mathbb{N} \) as:

\[ \phi^+_{|s|}(s) = \pi((|s| - 1), \sigma_{|s|}(s) - 1) \]  

(20)

Note that both \( \pi \) and \( \sigma \) are computable bijections. When we have the set \( s \) we can compute its cardinality \( |s| \) in linear time and compute \( \sigma_{|s|}(s) \) from \( s \) in polynomial time. When we have \( \phi^+_{|s|}(s) \) we can compute \( |s| \) and compute \( s \) from \( \sigma_{|s|}(s) \) in polynomial time. □

An elaborate example of the computation both ways is given in the appendix in paragraph 9. An example of the mapping can be seen in figure 13 under the header Cardinality Grid.

Note that in this proof the combinatorial number systems are defined on \( \mathbb{N}^+ \), while \( \pi \) is defined on \( \mathbb{N}^2 \). The results can easily be normalized by linear time computable translations: \( s^+ = (s_k, \ldots, s_2, s_1) \in \mathbb{N}^{+k} \to s = (s_k - 1, \ldots, s_2 - 1, s_1 - 1) \in \mathbb{N}^k \). For reasons of clarity, in the rest of the paper, we will ignore such corrections for \( \mathbb{N}^+ \) and use the function:

\[ \phi_{|s|}(s) = \pi((|s|), \sigma_{|s|}(s)) \]  

(21)

The construction of the proof of theorem 2 separates the set \( \mathcal{P}(\mathbb{N}) \) in an infinite number of infinite countable partitions ordered in two dimensions: in the columns we find elements with the same cardinality, in the rows we have the elements with the same index.

We give some examples. Consider the four tables in figure 13 with the following explanation:

1. The first table is a fragment of the simple Cantor function.
Fig. 13. This figure illustrates the effects of two specific injections (based on sum and product) of the form $\phi_\zeta(s) = \pi((\zeta(s), \theta_\zeta(s)))$ (equation 22). First table: basic Cantor grid. Second table: Mapping to finite sets of numbers. Third table: effect of the sum translation. Fourth table: effect of the product translation.

2. The second table illustrates the lexicographic ordering of finite sets of numbers in $\mathcal{P}(\mathbb{N})$ according to $\phi_{||}(s) = \pi((|s|), \sigma_{|s|}(s))$ (equation 21). Note the fact that we find the set $\{0, 1, \ldots, k\}$ on the line $y = 0$ in column $k$.

3. The third table computes the shift for the sum function $\phi_\Sigma(s) = (\Sigma s, \theta_{\Sigma s}(s))$.

4. The fourth table computes the shift for multiplication $\phi_H(s) = \pi(\Pi s, \theta_{\Pi s}(s))$.

4.6 A general theory of planar elastic translations for arithmetical functions

This suggests a general construction for the study of the information efficiency of arithmetical functions:

**Definition 10.** $\phi : \mathcal{P}(\mathbb{N}) \rightarrow \mathbb{N}$, a injection sorted on $\zeta$, is a mapping of the form:

$$\phi_\zeta(s) = \pi(\zeta(s), \theta_\zeta(s))$$

where $\pi$ is the Cantor function and:

- $\zeta : \mathcal{P}(\mathbb{N}) \rightarrow \mathbb{N}$ is a general arithmetical function operating on finite sets of numbers. It can be interpreted as a type assignment function that assigns the elements of $\mathcal{P}(\mathbb{N})$ to a type (column, sort) represented as a natural number.
– $\theta_k : \mathcal{P}(\mathbb{N}) \to \mathbb{N}$ is an index function for each type $k$, that assigns an index to the set in column $k$. The equation $\theta_{\zeta(s)}(s) = n$ should be read as: $s$ is the $n$-th set for which $\zeta(s) = k$.

By theorem 2 we have that $\phi||$ is efficiently countable. We can use $\phi||$ as a calibration device to evaluate $\phi\zeta$. If $\phi\zeta$ is a sorted injection the following mappings exists $\phi||^{-1} : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ and $\phi\zeta^{-1} : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ such that $\phi||^{-1}(\phi||)(x)), \phi\zeta(\phi\zeta^{-1}(x))$ are identities in $\mathcal{P}(\mathbb{N})$. Given this interconnectedness we can always use $\phi||$ to construct $\phi\zeta$:

**Algorithm 1** Compute the sorted index function $\theta_{\zeta(s)}(s)$ using $\phi||$.

```
1: $k \leftarrow 0$
2: $i \leftarrow 0$
3: while $k \leq \phi||(s)$ do
4:   if $\zeta(\phi||^{-1}(k)) = \zeta(s)$ then
5:     $k \leftarrow k + 1$
6:   else
7:     $i \leftarrow i + 1$
8:   end if
9: end while
10: $\theta_{\zeta(s)}(s) = i$
```

**Theorem 3.** If the function $\zeta$ exists and can be computed in polynomial time then sorted injections $\phi\zeta : \mathcal{P}(\mathbb{N}) \to \mathbb{N}$ of the form $\phi\zeta(s) = \pi(\zeta(s), \theta_{\zeta(s)}(s))$ exist and can be computed in time exponential to the representation of $s$.

Proof: We have to compute $\phi\zeta(s) = \pi(\zeta(s), \theta_{\zeta(s)}(s))$. The functions $\pi$ and $\zeta$ can be computed in polynomial time. The function $\theta$ can be computed using $\phi||$ with algorithm 1. This algorithm runs in time exponential in the representation of $k$ which is the index of the set $s$: $\phi||^{-1}(k) = s$. □

Note that algorithm 1 is a counting algorithm as discussed in paragraph 2.3 and that $\pi(\zeta(s), \theta_{\zeta(s)}(s))$ is a unique description of $s$ as referred to in our central research question 1: Are there uniquely identifying descriptions of objects that contain more information than names of the objects they denote? Observe that this procedure is a meta-algorithm. It abstracts completely from the semantics of the function $\zeta$.

There is a spectrum of planar translations as is illustrated by table 1. The interpretation of this table is as follows:

1. The first column gives the definition of the elastic function classes with a planar representation with increasing power:
   (a) The Cantor function with density 1 in $\mathbb{N}^2$.
   (b) Linear translations by a factor $c$. 

---

In the context of the algorithm, the notation $\mathcal{P}(\mathbb{N})$ represents the power set of natural numbers, and $\phi||$ is an indicator function that maps natural numbers to the power set, efficiently countable. The index function $\theta_k$ assigns an index to each set in column $k$. The equation $\theta_{\zeta(s)}(s) = n$ indicates that for a set $s$, $n$ is the position in the sequence of all sets for which $\zeta(s) = k$. The theorem formalizes the existence of sorted injections $\phi\zeta$ and their computability under certain conditions.
Polynomial translations by $cx^k$.

Sum translations, based on the sum of the set of natural numbers associated with the cell.

Product translations, based on the product of the set of natural numbers associated with the cell.

The Trivial translation defined by the function $\pi(x, y) = (\pi(x, y), 0)$ that projects all the cells of the Cantor space on the line $y = 0$.

2. The second column gives the functions to compute the information efficiency of the translation over the $x$-axis.

3. The third column gives the number of resulting different efficiency functions in the limit. Constant for linear transformations. Growing unboundedly for polynomial translations, to a different efficiency function for each cell for the Sum and the Product translations. For the trivial translation $\pi$ defines itself as its own efficiency function.

4. The fourth column gives the number of different efficiency functions per cell. Only for the Sum and the Product translations this becomes larger than 1.

5. The fifth column gives the resulting density function for $\mathbb{N}^2$ over the counter diagonal. Up to polynomial translations the density is 1. For Sum and Product translations the bijection becomes an injection, because there are only a finite number of sets that add up to, or multiply up to, a given number $n$. For the sum translation the density over the counter diagonal is roughly $x - \log x$ in the limit, for the product translation the same density becomes logarithmic. For the trivial translation it becomes 0.

| Translation over $\mathbb{N}^2$ | $\delta$ over $x$-axis | # Efficiency Functions | # per cell | Density in $\mathbb{N}^2$ |
|-------------------------------|------------------------|------------------------|------------|-------------------------|
| $\phi_1 = \pi$                | $\delta(1)$            | 1                      | 1          | 1                       |
| $\phi_c = \pi_c$              | $\delta(c)$            | $c$                    | 1          | 1                       |
| $\phi_{cx^k} = \pi_{cx^k}$    | $\delta(cx^k)$         | $\aleph_0$             | 1          | 1                       |
| $\phi_\Sigma$                 | $\delta(\Sigma)$      | $\aleph_0$             | $>2^x \approx x - \log x$ |
| $\phi_H$                      | $\delta(H)$            | $\aleph_0$             | $\geq 1$   | $\approx \log x$       |
| $\phi_\pi$                    | $\delta(\pi)$          | 1                      | 1          | $\approx 0$             |

Table 1. An overview of the spectrum of Cantor planar functions of increasing power.

Metaphorically one would envisage the spectrum in terms of a rubber sheet lying over the space $\mathbb{N}^2$. We measure the information efficiency resulting from the translations. When one starts to pull the sheet over the $x$-axis it starts to shrink over the $y$-axis. As long as one pulls with linear force the notion of a smooth information efficiency associated with a computable bijection over the space is conserved. When we pull with monotonely increasing force of a polynomial function, a computable bijection is still available, but the space starts to fluctuate...
with increasing discontinuities and the tension in the limit is infinite. When one
pulls superpolynomially, using functions on sets of numbers, the rubber sheet
starts to disintegrate at finite dimensions and stops to be a computable bijection.
It becomes an injection that is only occasionally locally computable on a point
to point basis, although every stage of the translation can be reconstructed from
the origin in exponential time. When we pull the sheet super exponentially the
shift is so big that the place of the image on the x-axis starts to carry information
about the origin of all sets for which the image is on the line \( x = c \). This is the
case for primes and square numbers that have only two generating sets. When
all points in the space \( \mathbb{N}^2 \) are mapped densely to the line \( y = 0 \) the translation
becomes the trivial inverse of the Cantor packing function, which is by definition
information efficient again.

4.7 A formal analysis of the objects \( \phi_\Sigma \) and \( \phi_\Pi \)

Define \( \Sigma s = \sum_{s_i \in s} s_i \) as the addition function for sets. Suppose \( s \) is the set
associated with the index \( (x, y) \) in \( \mathbb{N}^2 \) and \( \pi(x, y) \) in \( \mathbb{N} \). We have the sum function
for sets: \( \Sigma : \mathcal{P}(\mathbb{N}) \to \mathbb{N} \). The function \( \epsilon_\Sigma : \mathbb{N}^2 \to \mathbb{N}^2 \) defines an *elastic translation*
by a function \( \Sigma \) of the form:

\[
\epsilon_\Sigma(x, y) = (\Sigma s, \theta_\Sigma s(s))
\]

Here \( \theta_\Sigma : \mathcal{P}(\mathbb{N}) \to \mathbb{N} \) is the function that enumerates sets with the same sum.
We have the functions: \( \pi : \mathbb{N}^2 \to \mathbb{N} \) and \( \phi_\Pi : \mathcal{P}(\mathbb{N}) \to \mathbb{N} \). We define an injection
\( \phi_\Sigma : \mathcal{P}(\mathbb{N}) \to \mathbb{N} \) sorted on sum:

\[
\phi_\Pi(\epsilon_\Sigma(s)) = \phi_\Sigma(s) = \pi(\Sigma s, \theta_\Sigma s(s))
\]

A fragment of the function is shown in figure 13 under the heading Sum Grid.
The associated injection on \( \mathbb{N} \) is:

\[
\mathbb{N} \xrightarrow{\pi^{-1}} \mathbb{N}^2 \xrightarrow{\epsilon_\Sigma} \mathbb{N}^2 \xrightarrow{\pi} \mathbb{N}
\]

Observe that the shift is an injection. For every natural number there is only
a finite number of sets that add up to this number. The injection is dense when
sampled over the counter diagonal.

There are for each set \( s_k = \{1, 2, \ldots, k\} \), with \( \sum_{x \in s_k} x = u \), according to
equation 33 exactly \( B_k \) partitions that add up to \( u \). This means that there is a
super exponential number of sets that add up to \( u \) which gives, when sampled
over the counter diagonal, using \( \pi \), a density of 1 in the limit. This fact is re-
markable, at every line \( x = c \) we compress an infinite set to a finite one, but the
density of the resulting index stays close to \( x - \log x \).

In figure 14, we give a schematic overview of the translation \( \phi_\Sigma \) driven by the
Cantor function for finite sets \( \phi_\Pi \) at a finite moment \( t \). The area \((o, a, b)\) is the
Fig. 14. Schematic overview of the translation $\phi_\Sigma$ generated by $\phi_{||}$ at a finite moment $t$.

set of points that have been “surveyed” by $\phi_{||}$ up till now. The area below the red line illustrates the points that have been generated by $\phi_\Sigma$. The exponential function illustrated by the line $(0, c, d)$ marks the empty area where $\phi_\Sigma$ will never generate images. Point $(a, 0)$ is on line $y = 0$. It codes the set $\{0, 1, 2, \ldots a\}$ with the image $a'$ at $(\frac{1}{2}a(a - 1), 0)$. Point $a - 1'$ is the image of $(a - 1, 0)$ at $(\frac{1}{2}(a - 1)(a - 2), 0)$. The gaussian distribution $e$ illustrates the density of the points around $\frac{1}{2}a$ that corresponds to sets of numbers, generated by a random selection of $\frac{1}{2}a$ elements from the set $\{0, 1, 2, \ldots a\}$ that will have an image in the neighborhood of $(\frac{1}{2}(\frac{1}{2}a(\frac{1}{2}a - 1)), 0)$. Note that only a tiny fragment of these sets has been surveyed by $\phi_{||}$. About all others we have no information at time $t$.

Observe that figure 14 describes the situation of the computation of $\phi_\Sigma(s)$ at any finite moment in time and that the line $(a, b)$ is a hard information boundary. At any moment in time $t$ the characterization of the set $\phi_\Sigma$ by any finite segment of $\mathbb{N}$ is fundamentally incomplete. The images on the line $y = 0$ form a discontinuous cloud of points. There is an exponential amount of relevant computations that we have not seen yet. We propose to call such sets Semi-Countable. The name is motivated by the fact that these sets seem to occupy a place between countable and uncountable sets. There are no effective search procedures for such sets. These observations are formalized in the following theorem:

**Theorem 4.** The function $\phi_\Sigma$ can: 1) be computed by a recursive function in time exponential in the representation of $s$ but 2) not in polynomial time. 

Proof:
1. $\phi_\Sigma(s)$ has a finite definition and for every set $s$ can be generated in time exponential to the representation of the Cantor index $\phi_\eta(s)$ according to theorem 3.

2. Suppose that there is an algorithm $p$ that computes $\phi_\Sigma(s)$ in polynomial time. Observe that $p$ is either primitive recursive or $\mu$-recursive:
   - Algorithm $p$ cannot be a single primitive recursive function (constructive algorithm). This is an immediate consequence of the principle of characteristic information efficiency (see definition 6). There are two different grounds:
     - The expression with a free variable $x$:
       $$\phi_\Sigma(x) = \pi(\Sigma x, \theta_{\Sigma x}(x))$$
       is a function scheme. Each cell in $\mathbb{N}^2$ contains a different set $s$. Consequently $\phi_\Sigma(x)$ is a different primitive recursive computation for each $x = s$. Since $\Sigma s = k$ is information discarding the information about the original computation is lost. In fact: since for every set $s$ the computation $\phi_\Sigma(x) = n$ is different, the number $\phi_\eta(s) = \pi(x)$ is an index of the computation. Consequently we can only compute $\phi_\Sigma^{-1}(n) = s$, if we already know $s$.
     - The expression with a bound variable:
       $$\phi_\Sigma(s) = \pi(\Sigma s, \theta_{\Sigma s})$$
       is also a function scheme. Suppose $\Sigma s = u$ and $\theta_{\Sigma s} = v$. The expression $(\Sigma s, \theta_{\Sigma s})$, is a unique description of $s$: “$s$ is the $v$-th set that adds up to $u$”. By lemma 5 the information efficiency of this expression is not defined. This implies that the description is ad hoc: it identifies $s$ uniquely, but it is not clear how much information we have when we see it. By equation 36 the number of possible different primitive recursive computations for $\Sigma$ is super-exponential and generates a cloud of different values for the information efficiency of each set.
   - Algorithm $p$ cannot be a $\mu$-recursive function (search algorithm): The cell $(k, 0)$ is projected on cell $(\frac{1}{2}k(k + 1), 0)$, for all cells above $(k, 0)$ the shift is larger. This means that the boundary conditions of theorem 1 apply. $\phi_\Sigma$ generates an unbounded amount of information in the limit by equation 18. A polynomial time search algorithm can only generate a constant amount of information.

This implies that the denotation of the expression the $v$-th set that adds up to $u$” is dependent on our search method. The search method itself creates the information about the objects it finds. Consequently there is no best or most effective way to search for the $v$-th set that adds up to $u$”. This holds specifically for the first set that adds up to $u$”. The set $\phi_\Sigma$ cannot be searched effectively.
The fact that the sum operation is information discarding illustrates the difference with the situation of theorem 1. This is clarified by an analysis of the product translation. Observe that the product function is information efficient. We do not lose information when we multiply. This is an indication that $\phi_{\Pi}$ as a bijection is easier to compute than $\phi_{\Sigma}$. Define $\Pi s = \Pi_{s_i \in s} s_i$ as the multiplication function for sets. We define an injection $\phi_{\Pi}: P(\mathbb{N}) \rightarrow \mathbb{N}$ sorted on product:

$$\phi_{\Pi}(s) = \pi(\Pi s, \theta_{\Pi s}(s)) \quad (27)$$

A fragment of the function is shown in figure 13 under the heading Product Grid. A schematic analysis of the translation is given in figure 15. Observe that this shift is also an injection. The complexity of $\phi_{\Pi}$ is fundamentally different from the complexity of $\phi_{\Sigma}$. We analyze the conditions of the proof of theorem 4 in the context of $\phi_{\Pi}(s)$:

$\phi_{\Pi}(s)$ has a finite definition and for every set $s$ can be generated in time exponential to the representation of the Cantor index $\phi_{||}(s)$ according to theorem 3.

Since multiplication is information efficient, we cannot invoke the principle of characteristic information efficiency (see definition 6) to prove that the is no recursive function to compute it. The information about the original computation is kept in so far that the set of factors of $k$ always can be reconstructed.

For every natural number there is only a finite number of sets that multiply up to this number. There are for each set $s_k = \{1, 2, \ldots, k\}$, with $\Pi_{x \in s_k} x = k!$, according to equation 33 exactly $B_k$ partitions that multiply up to $k!$. This gives a density of the counter diagonal that is approximately logarithmic.
Since for every set $s$ the computation $\phi_H(x) = n$ is different, the number $\phi_H^{-1}(n) = s$, if we already know $s$. This is exactly the case for prime numbers, they code the information about their own multiplicative history. Since we know that primality is in $P$ the object $\phi_H$ is a bijection that at least partly for certain columns $x = c$ can be computed in polynomial time both ways.

4.8 The Subset Sum problem

We give a detailed analysis of the conceptualization of the Subset Sum problem using $\phi_\Sigma$. We get all possible instances of the subset sum problem when we generalize the sum operation over all possible subsets of $\mathbb{N}$. Let $f$ be the characteristic function of an infinite set $S \subseteq \mathbb{N}$. Example: the set $\{2, 5, 100, \ldots\}$ is coded as $f(1) = 2, f(2) = 5, f(3) = 100, \ldots$ Define $\Sigma_f(s) = \sum_{s_i \in s} f(s_i)$ as the addition function for sets of elements of $S$ ordered by their indexes in $\mathbb{N}$, e.g. $\Sigma_f(\{2, 3\}) = f(2) + f(3) = 105$. We define an injection $\phi_{\Sigma,f}: \mathcal{P}(\mathbb{N}) \rightarrow \mathbb{N}$ sorted on sum of $S$:

$$\phi_{\Sigma,f}(s) = \pi(\Sigma_f(s), \theta_{\Sigma_f(s)}s)$$

Such injections generate gaps on the $x$-axis such that no point in the space $\mathbb{N}^2$ has an image on the line $x = c$. In these cases the existence of such an image becomes a decision problem. Such decision problems can be computed in exponential time, but not in polynomial time:

**Lemma 4.** $\phi_{\Sigma,f}(s)$ can: 1) be computed in time exponential in the representation of $s$ but 2) not in polynomial time.

Proof: Suppose $\phi_{\Sigma,f}(s)$ can be solved in polynomial time. Observe that it is a generalization of $\phi_\Sigma(s)$. We could solve $\phi_\Sigma(s)$ by choosing a specific $f$ and solve it. This contradicts theorem 4. □

5 Conclusion

The objects $\phi_\Sigma$ and $\phi_H$ belong to the most complex constructions we will ever encounter in mathematics. $\phi_\Sigma$ encodes everything there is to know about addition, while $\phi_H$ encodes all we can ever know about multiplication including the locations of the primes. The class $\phi_H$ is easier to compute than $\phi_\Sigma$. The last class is exactly in the sweet spot of elastic planar translation where the tension between compression (information loss on the $y$-axis) and expansion (information generation over the $x$-axis) is maximal. Observe that any polynomial transformation of $\phi_\Sigma$ conserves the conditions of theorem 4. Only when we apply exponential transformations, the deformations become so extreme that the translations start to carry information about the original cells in their new location on the $x$-axis and we come close to the easier class $\phi_H$.

We mention some remarkable observations that could motivate further study of the objects $\phi_\Sigma$ and $\phi_H$:
The Cantor function maps \( \mathbb{N} \) onto \( \mathbb{N}^2 \). The objects \( \phi_\Sigma \) and \( \phi_\Pi \) correspond to an infinite compression of \( \mathbb{N}^2 \) over the \( y \)-axis that still has a dense representation in \( \mathbb{N} \) when "sampled" via the Cantor function.

In terms of Kolmogorov complexity this implies that \( \mathbb{N} \) is mapped onto itself without the expansion that normally is generated by injective recursive functions like \( y = x^2 \) or \( y = 2^x \). The images of the translations in \( \mathbb{N} \) generated by \( \phi_\Sigma \) and \( \phi_\Pi \) are incompressible numbers.

Consider figures 10, 14 and 15. Note that all natural numbers and measurements in these schematic geometrical images have logarithmic representations. Consequently any measurable neighborhood \( \epsilon > 0 \) has exponential size in the limit. We cannot search any area or distance with measure \( \epsilon > 0 \) efficiently in the limit.

By the same reasoning any measurable neighborhood that is more than \( \epsilon > 0 \) removed from \( x = 0 \) or \( y = 0 \) will contain an exponential number of cells with incompressible indexes in the limit.

Observe figures 14 and 15. At any finite stage of the construction of these images there is an exponential amount of points on the line \( y = 0 \) for which the corresponding image in the domain has not been found. These cells correspond to the descriptions "the first set \( x \) that adds/multiplies up to \( k \).

The semi-countable sets form an natural link between the countable and uncountable sets: \( \mathbb{N} \) is countable, \( \mathcal{P}(\mathbb{N}) \) is uncountable. The class \( \mathcal{P}(\mathbb{N}) \) could be seen as the class that stretches countability to the maximum. The elaborate scheme to map the class \( \mathcal{P}(\mathbb{N}) \) to \( \mathbb{N} \) via the Cantor function and combinatorial number systems is in fact an application of the axiom of choice: we define an infinite set of different subclasses that allows us to make an infinite set of choices for the first element. If we then define super polynomial information generating operations on such a mapping, the fabric of countability is torn apart and we enter the universe of semi-countable sets.

6 Acknowledgements

I thank Peter van Emde Boas for many inspiring discussions and Rini Adriaans for continuous support. I thank the University of Amsterdam and especially Cees de Laat for allowing me the leeway to pursue my research interests.

References

1. P.W. Adriaans and P.E. Van Emde Boas. Information, computation and the arrow of time. In B. S. Cooper and A. Sorbi, editors, Computation and Logic in the Real World, COMPUTABILITY IN CONTEXT, chapter I. Imperial College Press, 2011.
2. R. Fueter and G. Pólya. Rationale Abzählung der Gitterpunkte. Vierteljschr. Naturforsch. Ges. Zürich, 58:280–386, 1923.
3. M. Li and P. Vitányi. An Introduction to Kolmogorov Complexity and Its Applications. Springer-Verlag, 2008.
4. P. Odifreddi and B.S. Cooper. Recursive functions. In E.N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2016 edition, 2016.

5. P.W. Adriaans. Information. In E.N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2018 edition, 2018.
7 Appendix: Combinatorial Considerations

The product of a set of numbers \( s \) is: \( \Pi_{i \in s} s_i \). The sum of a set of numbers \( s \) is: \( \Sigma_{i \in s} s_i \). The set of numbers \( \{1, 2, \ldots, n\} \) is called an initial segment of \( \mathbb{N} \) of size \( i \). The sum of an initial segment \( \{1, 2, \ldots, n\} \) is:

\[
\Sigma_{i=1}^{n} i = \frac{1}{2}n(n+1)
\]  

(29)

The product of an initial segment \( \{1, 2, \ldots, n\} \) is:

\[
\Pi_{i=1}^{n} i = n!
\]  

(30)

The number of ways to write down \( n \) pairs of balanced brackets is given by the Catalan number:

\[
C_n = \frac{1}{n-1} \binom{2n}{n}
\]  

(31)

The Stirling number of the second kind is the number of ways a set of \( n \) numbers can be partitioned into \( k \) non-empty subsets:

\[
\left\{ \begin{array}{l}
\{a\} \\
\{b\}
\end{array} \right\} = \frac{1}{k!} \sum_{i=0}^{k} (-1)^{k-i} \binom{k}{i} (k-i)^n
\]  

(32)

The Bell number counts the number of possible partitions of a set:

\[
B_n = \Sigma_{k=0}^{n} \left\{ \begin{array}{l}
\{n\} \\
\k
\end{array} \right\}
\]  

(33)

Let \( \oplus_{\varnothing, \varphi} : \mathbb{N}^2 \rightarrow \mathbb{N} \) be a tensor operator that is non-commutative and non-associative. Suppose \( \# \oplus_{\varnothing, \varphi} \) is the number of different computations we can perform with \( \oplus \) on a set of \( n \) numbers. It is given by the number of ways we can write \( n-1 \) pairs of brackets in \( n! \) different sequences of strings:

\[
\# \oplus_{\varnothing, \varphi} (n) = n! C_{n-1} = n! \frac{1}{n} \left( \frac{2(n-1)}{n-1} \right)
\]  

(34)

Suppose \( \oplus_{\varnothing, \varphi} \) is commutative and non-associative, then, to our knowledge, there is no formula that describes the number of different computations for \( n \) objects, but it is, for large enough \( n \), certainly bigger than the total number of unbracketed partitions, which is given by the Bell number, and thus smaller than \( \# \oplus_{\varnothing, \varphi} (n) \) but still super exponential:

\[
\# \oplus_{\varnothing, \varphi} (n) > \# \oplus_{\varnothing, \varphi} (n) > B_n > 2^n
\]  

(35)

If \( \oplus_{\varnothing, \varphi} \) is commutative and associative, then the number of different computations for \( n \) objects is 1: all computations can be transformed into all others. Which gives the following correlation between structural rules and number of computations defined on sets:

\[
\# \oplus_{\varnothing, \varphi} (n) = n! C_{n-1} > \# \oplus_{\varnothing, \varphi} (n) > B_n > 2^n > \# \oplus_{\varnothing, \varphi} (n) = 1
\]  

(36)
8 Appendix: Information Efficiency of Recursive Functions

We have defined \( \forall (x \in \mathbb{N}) I(x) = \log x \). Note that \( n \in \mathbb{N} \) is a cardinality and that the measurements \( I(n) \) represent a cloud of scalar point values in \( \mathbb{R} \). The relation between the two sets is described by the Taylor series for \( \log(x + 1) \):

\[
I(s + 1) = \log(x + 1) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^n}{n}
\]  

(37)

We have \( \lim_{x \to \infty} I(x + 1) - I(x) = 0 \). In the limit the production of information by counting processes stops as specified by the derivative \( \frac{d}{dx} \ln x = \frac{1}{x} \). The intuition that one could express based on this observation is that information in numbers in the limit becomes extremely dense, much denser than the values in ‘normal’ counting processes.

Given definition 5. We can construct a theory about the flow of information in computation. For primitive recursive functions we follow [4]. Given \( \log 0 = 0 \) the numbers 0 and 1 contain no information.

- **Composition of functions** is information neutral:
  \[
  \delta(f(g(x))) + \delta(g(x)) = \log f(g(x)) - \log g(x) + \log g(x) - \log x = \log f(g(x)) - \log x
  \]

- The constant function \( z(n) = 0 \) carries no information \( z^{-1}z(n) = \mathbb{N} \).
- The first application of the successor function \( s(n) = x + 1 \) is information conserving:
  \[
  \delta(s(0)) = \log(0 + 1) - \log 0 = 0
  \]

- The successor function expands information for values > 1. By equation 37 we have:
  \[
  I(s) = \log(x + 1) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^n}{n} > \log x
  \]

Consequently:

\[
\delta(s(x)) = I(s(x)) - \log x - \log 1 = \log(x + 1) - \log x = \epsilon > 0
\]

Note that \( \epsilon \) goes to zero as \( x \) goes to \( \infty \).

- The projection function \( P_{i,n}((x_1, x_2, \ldots, x_n) = x_i \), which returns the i-th argument \( x_i \), is information discarding. Note that the combination of the index \( i \) and the ordered set \( (x_1, x_2, \ldots, x_n) \) already specifies \( x_i \) so:
  \[
  \delta(P_{i,n}(x_1, x_2, \ldots, x_n) = I(x_i) - \log i - I(x_1, x_2, \ldots, x_n) < 0
  \]

- **Substitution.** If \( g \) is a function of \( m \) arguments, and each of \( h_1, \ldots, h_m \) is a function of \( n \) arguments, then the function \( f \):
  \[
  f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n))
  \]
is definable by composition from $g$ and $h_1, \ldots, h_m$. We write $f = [g \circ h_1, \ldots, h_m]$, and in the simple case where $m = 1$ and $h_1$ is designated $h$, we write $f(x) = [g \circ h](x)$. Substitution is information neutral:

$$\delta(f(x_1, \ldots, x_n)) = \delta(g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n))) = I(f(x_1, \ldots, x_n)) - I(x_1, \ldots, x_n)$$

Where $I(f(x_1, \ldots, x_n))$ is dependent on $\delta(g)$ and $\delta(h)$. - Primitive Recursion. A function $f$ is definable by primitive recursion from $g$ and $h$ if $f(x, 0) = g(x)$ and $f(x, s(y)) = h(x, y, f(x, y))$. Primitive recursion is information neutral:

$$\delta(f(x, 0)) = \delta(g(x)) = I(g(x)) - I(x)$$

which dependent on $I(g(x))$ and

$$\delta(f(x, s(y))) = \delta(h(x, y, f(x, y))) = I(h(x, y, f(x, y))) - I(x) - I(y)$$

which is dependent on $I(h(x, y, f(x, y)))$.

Summarizing: the primitive recursive functions have one information expanding operation, counting, one information discarding operation, choosing, all the others are information neutral.

The information efficiency of more complex operations is defined by a combination of counting and choosing. We analyse the definition of addition by primitive recursion: suppose $f(x) = P_1^1(x) = x$ and $g(x, y, z) = S(P_2^1(x, y, z)) = S(y)$. Then $h(0, x) = x$ and $h(S(y), x) = g(y, h(y, x), x) = S(h(y, x))$. The information efficiency is dependent on the balance between choosing and counting in the definition of $g$:

$$\delta(g(x, y, z)) = I(S(y)) - I(x, y, z) = \left(\sum_{n=1}^{\infty} (-1)^{n-1} \frac{y^n}{n}\right) - I(x, y, z)$$

Note that the operation loses information and the information loss is asymmetrical. The commutativity and associativity of information efficiency for complex arithmetical operations are not trivial:

**Theorem 5 (Information Efficiency of Elementary Arithmetical Operations).**

- Addition of different variables is information discarding. In the case of addition we know the total number of times the successor operation has been applied to both elements of the domain: for the number $c$ our input is restricted to the tuples of numbers that satisfy the equation $a + b = c (a, b, c \in \mathbb{N})$. Addition is information discarding for numbers $> 2$:

$$\delta(x + y) = \log(x + y) - \log x - \log y < 0$$ (38)
Addition of the same variable has constant information. It measures the reduction of information in the input of the function as a constant term:

\[ \forall(x)\delta(x + x) = \log 2x - \log x = \log 2 \]  

(39)

Commutativity for addition is information efficient:

\[ \delta(x+y) = \log(x+y) - \log x - \log y = \log(y+x) - \log y - \log x = \delta(y+x) \]  

(40)

Associativity for addition is not information efficient:

\[ \delta(x + (y + z)) = \log((x + y) + z) - \log x - \log(y + z) \neq \log((y + z) + x) - \log(x + y) - \log z = \delta((x + y) + z) \]  

(41)

Multiplication of different variables is information conserving. In the case of multiplication \(a \times b = c\) (\(a, b, c \in \mathbb{N}\)) the set of tuples that satisfy this equation is much smaller than for addition and thus we can say that multiplication carries more information. If \(a = b\) prime (excluding the number 1) then the equation even identifies the tuple. Multiplication is information conserving for numbers \(> 2\):

\[ \delta(x \times y) = \log(x \times y) - \log x - \log y = 0 \]  

(42)

Multiplication by the same variable is information expanding. It measures the reduction of information in the input of the function as a logarithmic term:

\[ \forall(x)\delta(x \times x) = \log(x \times x) - \log x = \log x > 0 \]  

(43)

Commutative for multiplication is information efficient:

\[ \delta(x \times y) = \log(x \times y) - \log x - \log y = \log(y \times x) - \log y - \log x = \delta(y \times x) \]  

(44)

Associativity for multiplication is information efficient:

\[ \delta(x \times (y \times z)) = \log((x \times (y \times z)) - \log x - \log(y \times z) = \log((x \times y) + z) - \log(x \times y) - \log z = \delta((x \times y) \times z) \]  

(45)

Estimating the information efficiency of elementary functions is not trivial. From an information efficiency point of view the elementary arithmetical functions are complex families of functions that describe computations with the same outcome, but with different computational histories. For addition the effect is significant:

Lemma 5. If \(s\) is a set of natural numbers then \(\delta(\Sigma_{i \in s}s_i)\) is not defined.

Proof: Immediate consequence of theorem 5 the information efficiency of addition is non-associative. By equation 36 the number of possible different computations is super-exponential in the cardinality of the set and generates a cloud of different values for the information efficiency of each set. □
As an illustration we work out the example given in the introduction (see par. 1.1). Suppose we want to add the numbers 2, 47, 53 and 98. Most of us will see the pattern that makes this easy: \((2 + 98) + (47 + 53) = 100 + 100 = 200\). The numbers have a special relationship that makes the result less surprising, and therefore less informative. Apart from that the conditional information in the number 200 given the set \(\{2, 47, 53, 98\}\) varies fundamentally with the method of computation:

- As a composed function of 4 variables:
  
  \[
  2 + 47 + 53 + 98 = 200 \\
  \delta(2 + 47 + 53 + 98) = \\
  \log 200 - \log 2 - \log 47 - \log 53 - \log 98 \approx 7.64 - 15.56 - 6.64 - 6.61 = \\
  \text{Total: } -12.17
  \]

- Storing intermediate results, method 1:
  
  \[
  (2 + 47) + (53 + 98) = 200 \\
  \delta(2 + 47) = \log 49 - \log 2 - \log 47 \approx 5.61 - 15.56 = -0.95 \\
  \delta(53 + 98) = \log 151 - \log 53 - \log 98 \approx 7.245, 565, 78 = -4, 1 \\
  \delta(49 + 151) = \log 200 - \log 49 - \log 151 \approx 7, 645, 617, 24 = -5, 21 \\
  -0, 95 - 4, 1 - 5, 21 = \\
  \text{Total: } -10, 26
  \]

- Storing intermediate results, method 2:
  
  \[
  (2 + 98) + (47 + 53) = 200 \\
  \delta(2 + 98) = \log 100 - \log 2 - \log 98 \approx 6.64 - 1 - 6.61 = -0.97 \\
  \delta(47 + 53) = \log 100 - \log 47 - \log 53 \approx 6.64 - 5.56 - 5.78 = 4, 70 \\
  \delta(100 + 100) = \log 200 - \log 100 - \log 100 \approx 7, 64 - 6, 64 - 6, 64 = -5, 64 \\
  -0.97 - 4.70 - 5.64 = \\
  \text{Total: } -11, 31
  \]

- Storing intermediate results, method 2, and applying equation 39:
  
  \[
  \delta(100 + 100) = \log_2 2 = 1 \\
  \text{Total: } -0.97 - 4.70 + 1 = -4.67
  \]
Observe that basic composed equation 46 is not the most information efficient one with value $-12, 17$. Storing intermediate results actually improves the information efficiency in equations 47 with value $-10, 26$ and 49 with value $-11, 31$. The latter is somewhat less efficient, a situation that changes when we apply equation 39.

The associativity of information efficiency for multiplication is an ‘accident’ caused by the information conservation of the operation, but the effect of collapse of arguments as in equation 43 remains. Summarizing: some arithmetical operations expand information, some have constant information and some discard information. The flow of information is determined by the succession of types of operations, and by the balance between the complexity of the operations and the number of variables.

9 Appendix: Computing the bijection between $\mathcal{P}(\mathbb{N})$ and $\mathbb{N}$ via $\mathbb{N}^2$ using the Cantor packing function: an example

9.1 From $\mathcal{P}(\mathbb{N})$ to $\mathbb{N}$

Take a random finite set of natural numbers:

$$\{0, 3, 5, 7, 9, 10\}$$

Convert to $\mathbb{N}^+$:

$$1, 4, 6, 8, 10, 11$$

Compute the index in a combinatorial number system of order 6

$$\sigma_6(s) = \binom{11}{6} + \binom{10}{5} + \binom{8}{4} + \binom{6}{3} + \binom{4}{2} + \binom{1}{1} =$$

$$462 + 252 + 70 + 20 + 6 + 1 = 811$$

Compute the Cantor index for location $(6, 811)$

$$\pi(6, 811) = 1/2((6 + 811 + 1)(6 + 811)) + 811 = 334964$$

9.2 From $\mathbb{N}$ to $\mathcal{P}(\mathbb{N})$

Take the index: $\pi(x, y) = 334964 = z$. 
Compute the location for 334964

$$\pi(x, y) = \frac{1}{2}(x + y + 1)(x + y) + y$$

Define 2

$$w = x + y$$

$$t = \frac{w(w + 1)}{2} = \frac{w^2 + w}{2}$$

$$z = t + y$$

Here $t$ is the triangular number of $w$. We solve:

$$w^2 + w - 2t = 0$$

$$w = \frac{\sqrt{8t + 1} - 1}{2}$$

which is a strictly increasing and continuous function when $t$ is a non-negative real. Since

$$t \leq z = t + y < t + (w + 1) = \frac{(w + 1)^2 + (w + 1)}{2}$$

we get that

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

and thus

$$w = \left\lfloor \frac{\sqrt{8 \times 334964} + 1 - 1}{2} \right\rfloor = 817$$

$$t = (817^2 + 817)/2 = 334153$$

$$y = zt$$

$$y = 334964 - 334153 = 811$$

$$x = 817 - 811 = 6$$

The set has cardinality 6 and index 811.

2 We follow the computation given in https://en.wikipedia.org/wiki/Pairing_function, retrieved February 27, 2018.
Compute the set associated with location \((6, 811)\) We reconstruct the number sequence from \(\mathbb{N}^+\). Compute: \(\binom{11}{6} < 811 = 11.802\ldots\). The biggest number is 11

\[ 811 - \binom{11}{6} = 349 \]

Compute: \(\binom{10}{5} < 349 = 10.521\ldots\)
Next number is 10.

\[ 349 - \binom{10}{5} = 97 \]

\[ 97 - \binom{8}{4} = 27 \]

\[ 27 - \binom{6}{3} = 7 \]

\[ 7 - \binom{4}{2} = 1 \]

\[ \binom{1}{1} = 1 \]

The sequence is 1, 4, 6, 8, 10, 11. Convert to \(\mathbb{N}\). The original set is \(\{0, 3, 5, 7, 9, 10\}\)