Utrecht University

Faculty of Humanities

Artificial Intelligence

A Transfinite Complete One-Way Function

Author:
Kim Beelen
5902878

Supervisor:
Dr. Benjamin Rin

Second evaluator:
Dr. Natasha Alechina

Bachelor's Thesis, 15 ECTs

May 5, 2021

Abstract

Infinite time Turing machines and ordinal Turing machines are used to perform transfinite computational tasks. In order to further explore the capabilities of these models, this research looks into one-way functions and the possibility to generalize them to work in a transfinitary context. One of the greatest problems in (finitary) computer science is determining whether these functions exist or not. Crucially, it bears mentioning that the notion of one-way function is probabilistic in nature. In 2003, Levin presented a complete one-way function—a function guaranteed to be one-way if any one-way functions exist—based on the tiling problem. His findings were further developed in research by Kozhevnikov and Nikolenko in 2009. Following their research, we define a notion of transfinite one-way function, for any suitable generalized system of probability. Such a notion has not been defined before, and the present paper provides the groundwork needed to do so. To this end, we consider one possible candidate example of a generalized system of probability sufficient to ground the notion of transfinite one-way function—namely, Benci's theory of non-Archimedean probability—together with the theory of surreal numbers. Our main result proves that every ordinal polynomial-time computation on an ordinal Turing machine can be simulated by a transfinite tiling problem, and that transfinite tiling is therefore a transfinite complete one-way function.

Keywords: *Turing machine, infinite time Turing machine, ordinal Turing machine, transfinite computation, transfinite probability, one-way function, tiling problem.*

Acknowledgments

I would like to express my gratitude to Dr. Benjamin Rin for his useful guidance throughout the process of writing this thesis, and for his enthusiasm for the subject. I am also thankful to Dr. Lorenzo Galeotti for his helpful suggestions and criticism. I thank Dr. Natasha Alechina as well for her insightful corrections and feedback.

Contents

Chapter 1

Introduction

The Church-Turing thesis tells us that a function can be computed by an effective algorithm, if and only if it can be computed by a Turing machine. While the scope of all that we can compute is infinitely large, we are limited by the fact that a classic Turing machine (TM) can only successfully perform computations of finite length. In order to perform supertasks, which are computational tasks involving infinitely many steps, a more powerful model of computation is required. Such a model was first introduced by Hamkins and Lewis in 2000 [17]. Their infinite time Turing machine (ITTM) model is able to perform computations of transfinite ordinal length. A few years later, the model was extended by Koepke to the so-called ordinal Turing machine (OTM) [22], which additionally has access to a computation space of transfinite ordinal length.

In the years following the work of Hamkins and Lewis, a sizeable number of further papers have been published, see for example [5, 18, 32, 34]. However, there is still room for more research. This paper will focus on the concept of one-way functions, of which we are unsure whether they exist or not. A construction of a complete one-way function has been proposed by Levin [26], which was elaborated on by Kozhevnikov and Nikolenko [23]. It is the so-called *Tiling Expansion* function, which is based on the tiling problem [4]. Our work researches whether this function can be generalized to simulate an OTM. This will give us more insight on the potential of these powerful TM models.

The existence of one-way functions on OTMs would be useful in the field of cryptography. For instance, if Alice were to send a message to Bob, while making sure that the message cannot be intercepted by outsider Eve, Alice could encrypt the message using a one-way function. Then, it would be infeasible for Eve to retrieve the original message by computing the inverse of that function. In the finitary case, one differentiates between OWFs and functions that are not one-way. In the transfinitary case, it of course takes infinite time both to calculate the value and inverse of a function. However, if transfinite OWFs exist, then the infinitely many steps to compute the inverse will be much greater than to compute its value [6], resulting in a safe encryption algorithm. The field of transfinite cryptography is young, but promising. We refer to [33] for more.

The problem will be approached by answering the following research question: *Does a transfinite complete one-way function exist?* An answer to this question can be found by answering the following subquestions: *Can a probabilistic ordinal Turing machine be defined?* and *Can Tiling be generalized in such a manner that it becomes a transfinite complete one-way function that simulates ordinal Turing machines?*

This paper is structured as follows. Chapter 2 describes the classic Turing machine model that is used to perform finite computations. It also introduces the concept of one-way functions, along with an example of a complete one-way function. Chapter 3 focuses on transfinite computational models, namely the infinite time Turing machine and the ordinal Turing machine, and explains the concept of ordinal numbers. In chapter 4, the material from the previous two chapters is combined to design a transfinite complete one-way function. The conclusion and discussion are found in chapter 5.

Chapter 2

Finite computation

2.1 Classic Turing machines

In 1937, Alan Turing introduced the classic Turing machine model [36]. It is a model of computation that is able to perform a computation on the natural numbers if and only if it is describable by an algorithm, as stated by the Church-Turing thesis. A classic Turing machine has a working tape of infinite length in one direction, consisting of cells. The tape cells are indexed by the natural numbers, and the tape therefore has length equal to the cardinality of the set of natural numbers. The machine also has a read/write head, which sits on one cell at any given time. At any given moment, the cells of the tape have symbols on them. The string of symbols on the tape before the computation starts is called the input, the string of symbols on the tape at the end of the computation (if any) is called the output.

The formal definition of a Turing machine used in this work is based on the definition from [23]:

Definition 1. A Turing machine is an ordered 6-tuple, $(Q, \Sigma, \Gamma, \delta, s, h)$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. $\Sigma = \{0, 1\}$ is the input alphabet,
3. Γ is the tape alphabet, with $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$,
5. $s \in Q$ is the start state,
6. h is the halting state.

A Turing machine computes a function $f : \Sigma^* \to \Sigma^*$ if on every input w, it halts with just $f(w)$ on its tape [35, Definition 5.17], where Σ^* denotes the set of all strings that can be formed using symbols from set Σ. Initially, the input

is written onto the tape, followed by infinitely many blank symbols. At the start of the computation, the machine is in the start state and its head points to the first cell. The machine then follows the algorithmic steps of the program describing the function, and at each computation step it can read from and write onto the tape cell where the head is currently at. Depending on the program instructions, the head then moves to the left or right adjacent cell and can go into the next state. When the computation is finished, the machine enters the halting state. The tape then contains the output of the computation.

Certain algorithms cause the machine to perform looping instructions. While this does not necessarily cause any trouble, it is possible for a loop to run infinitely many times, which results in a computation of infinite length. In this situation, the machine never enters a halting state. That is, the computation never stops and gives an output. But can we determine whether it will halt eventually? Assuming the Church-Turing thesis is true, the undecidability of the halting problem means that no algorithm can reliably answer this question. Despite this problem, the Turing machine is a highly useful model in computability theory.

The Turing machine model can be altered without changing its computational strength by, for example, adding additional tapes and allowing the tape to be of infinite length in both directions. For these changes it is proven that they can be simulated by a normal single-tape TM that is equivalent [35, Theorem 3.13] [35, Exercise 3.18]. This also holds for TM models in which it is allowed to perform a computation step where the head remains positioned on the same cell, or ones in which the head's ability to move left is replaced by moving to the leftmost cell [35, Exercises 3.19-3.20].

An important distinction is that between deterministic and non-deterministic Turing machines. The classic TM model is deterministic. For these machines, there is exactly one transition in δ for each possible combination of states and symbols in $Q \times \Gamma$. Non-deterministic Turing machines (NTMs) are generalizations of deterministic Turing machines. They allow multiple or zero transition options per combination. As described by Kozhevnikov and Nikolenko in [23], an NTM is a regular deterministic TM with an additional tape containing a 'witness'. An NTM N computes a function $f : \Sigma^* \to \Sigma^*$ if for every input x there exists a witness $y \in \Sigma^*$ such that $N(x, y) = f(x)$, and for no witness the machine terminates with a wrong answer. The NTM may end without giving any answer. Every NTM has an equivalent deterministic TM [35, Theorem 3.16].

A special variant of non-deterministic Turing machine is the probabilistic Turing machine. Its computational behavior is the same as the behavior of NTMs as described above, but with a new notion of what it means to compute a function. Using the definition from Kozhevnikov and Nikolenko in [23], a probabilistic Turing machine is said to compute a function f on input x with probability p if

$$Pr_{y \in U_m}[M(x, y) = f(x)] = p,$$

where m is the number of random symbols from Σ (the witness length), U_m is

the uniform distribution on strings of length m over alphabet Σ, and $y \in U_m$ means "witness y is taken over the distribution U_m".

2.2 One-way functions

2.2.1 Computability and complexity

As was mentioned before, the Church-Turing thesis states that a function is effectively computable if and only if it is computable by a Turing machine. However, effectivity does not entail feasibility. Some computations can be computed efficiently, while others take too much memory or time to be realistically plausible. This distinction is of great use in cryptography. In cryptography, we wish to take advantage of functions that are easy to compute but hard to invert.

We say a function is *feasibly* computable if it can be computed in polynomial time. A function f is said to be computable in polynomial time if its running time is upper bounded by a polynomial in the size of the input n, where a polynomial is a term of the form $a_k n^k + a_{k-1} n^{k-1} + .. + a_2 n^2 + a_1 n + a_0$, where $a_1, ..., a_k$ are constants. Equivalently, we can simply write the bound as n^k, since n^k grows faster than any polynomial whose leading term's exponent is smaller than k. This formalization of the concept of feasibility is due to Cobham [9, 16].

Functions that are computable in polynomial time by a deterministic Turing machine, form the complexity class P.[1] If computing the inverse of f is not possible in polynomial time, then f is considered hard to invert. The problem of inverting a function that is indeed invertible, can be solved non-deterministically using a witness. Problems like this, that are solvable in polynomial time by an NTM, belong to the complexity class NP.

We know class P is a subset of class NP. However, we are unsure whether P and NP are actually equivalent or not. This question can be answered by finding a polynomial time algorithm that solves an NP-complete problem. A problem A in NP is NP-complete if every problem B in NP is polynomial time reducible to it. I.e., B is transformable into A using a polynomial time TM. Finding an NP-complete problem that is solvable in polynomial time would allow us to conclude that P = NP, since then every problem in NP must be in P. To this day, no such NP-complete problem has been found. This problem of determining P $\stackrel{?}{=}$ NP is widely known as the P versus NP problem, and most believe P $\neq$ NP to be true [14].

2.2.2 Strongly and weakly one-way functions

Functions such as f, that are feasibly computable and hard to invert, are called one-way functions (OWFs). A distinction is made between weakly and strongly

[1] Technically, this is actually the class FP, since the class P consists of decision problems. For simplicity we leave aside this distinction in the present discussion.

OWFs. This work uses the formal definitions from Kozhevnikov and Nikolenko's 2009 article [23].

Definition 2. A function $f : \{0,1\}^* \to \{0,1\}^*$ is called *strongly one-way* if the following conditions hold:

1. Feasible to compute: There exists a deterministic Turing machine M with input alphabet $\{0,1\}$ that computes f in polynomial time;

2. Hard to invert: For every probabilistic Turing machine M', every polynomial p, and every sufficiently large n,

$$Pr_{M',x\in U_n}[M'(f(x),1^n) \in f^{-1}(f(x))] < \frac{1}{p(n)},$$

where 1^n denotes n many 1s, and where the probability is taken over the random bits of M' and input x (both distributions are uniform).

That is, f is strongly one-way if there is no adversary that can invert it on any significant fraction of inputs.

Definition 3. A function $f : \{0,1\}^* \to \{0,1\}^*$ is called *weakly one-way* if the following conditions hold:

1. There exists a deterministic Turing machine M with input alphabet $\{0,1\}$ that computes f in polynomial time;

2. There exists a polynomial p such that for every probabilistic Turing machine M' and for every sufficiently large n,

$$Pr_{M',x\in U_n}[M'(f(x),1^n) \notin f^{-1}(f(x))] > \frac{1}{p(n)},$$

where 1^n denotes n many 1s, and where the probability is taken over the random bits of M' and input x (both distributions are uniform).

Thus, f is weakly one-way if for every adversary there is a significant fraction of inputs where it fails to invert f.

Like P versus NP, the question whether OWFs exist at all is an important problem that remains unsolved. If we were to know that OWFs exist, then this would also allow us to safely conclude a solution for the P versus NP problem, namely that P $\neq$ NP [15, page 92]. This must be true for OWFs to exist, since if we had P = NP, then computing a function with a deterministic polynomial time machine would be as complex as inverting it with a nondeterministic polynomial time machine. Note that this implication only goes one way: P $\neq$ NP is **not** known to imply the existence of one-way functions.

Ideally, we want to find a function that we can prove to be one-way. Multiplication of two large prime numbers is a good candidate for a one-way function. Inverting the multiplication of such numbers involves integer factorization: the

decomposition of a composite number into a product of smaller integers. The question of whether there exists a polynomial time algorithm for integer factorization is unsolved. If we were sure that there does not exist such a function, then multiplication of large primes would be one-way.

The current lack of knowledge on the existence of OWFs causes uncertainty in fields where multiplication and other OWF candidates are applied, such as the RSA algorithm in cryptography. If it were known that OWFs exist and multiplication is one such function, then this would imply that RSA is a safe encryption algorithm. Another application of one-way functions in cryptography is that of pseudorandom number generators, which are proven to exist if and only if OWFs exist [19]. Because of the bi-implication, a proof of existence of a pseudorandom number generator would directly imply the existence of OWFs. However, such a proof has not yet been given either. The existence of one-way functions thus remains unsolved to this day.

2.2.3 Complete one-way functions

While we do not know if one-way functions exist, the concept of complete OWFs might help us to learn more about these functions. The concept was introduced by Levin in 1987 [25]. With this notion, we can reason about OWFs without knowing if they exist. And using a concrete function that is proven to be complete one-way, the question of their existence is reduced to the question of whether this particular function is one-way.

We use the formal definition of a complete OWF from [23]:

Definition 4. Assume that a function $f : \{0,1\}^* \to \{0,1\}^*$ has the following property: if there exists a one-way function $g : \{0,1\}^* \to \{0,1\}^*$, then f is also one-way. Then f is called a complete one-way function.

Similar to the Cook-Levin theorem, which states that the Boolean satisfiability (SAT) problem is NP-complete [11, 24], Levin searched for a complete problem for one-way functions. As proposed by Levin, a complete problem, and thus a complete one-way function, can be constructed as a universal Turing machine whose computations can in turn be 'transformed into combinatorial objects'. For clarity: in the proof of the Cook-Levin theorem (for which one can refer to [35, Theorem 7.37]), these combinatorial objects appear as tableaus, which are tables that visualize the configurations of a Turing machine computation. The proof uses this construction to show that any language in NP reduces to it. Combined with proving that SAT itself is in NP, this provides a proof that SAT is indeed NP-complete. Levin suggested that a similar approach can be used to construct a complete one-way function [25, 26].

2.3 The Tiling simulating function

The first construction of a complete OWF was given by Levin in 1987 [25], but was not very natural. He presented a more useful construction in 2003, called

Tiling Expansion [26]. It is more natural in the sense that it does not explicitly use a Turing machine computation in its definition, but constructs a definition based on a TM computation. Tiling Expansion is a modification of the existing *tiling problem*, which is the problem of deciding whether a given set of tiles with colored edges can cover a certain grid, such that the colors of adjacent sides match [4]. This is also the core of Levin's Tiling Expansion.

Definition 5. *Tiling Expansion* is the following function: Given a set of tiles (called *permitted* tiles) and a row of tiles taken from this set, expand the row to a square in which the symbols match on adjacent tiles, then output the bottom row and the set of permitted tiles.

Tiling Expansion thus transforms an input string, which is put on the top row of the square in tile form, into an output string, which is the result on the bottom row of tiles, such that all tiles in between have matching adjacent edges.

In 2009, Kozhevnikov and Nikolenko gave an alternative presentation of Levin's results, with minor alterations [23]. It is this version that we present in what follows. Kozhevnikov and Nikolenko adjust Levin's Tiling Expansion definition and define it as a new problem called the *Tiling Simulating function*, or *Tiling*, for short. An instance of Tiling is an input string x, along with a finite set T of tiles. A *tile* is defined as a square whose edges are marked with a symbol from a finite set $\mathcal{A}$ of symbols. A *tiling* of an $n \times n$ grid is a set of n^2 tiles from T covering the grid such that symbols on adjacent edges match. Any tile from T may be used as many times as needed in the tiling. We say $x \xrightarrow{!}_T y$ if there exists a unique, length-preserving (i.e., $|x| = |y|$) tiling of an $|x| \times |x|$ grid that transforms x over $\mathcal{A}$ into y. That is, at each step in the tiling process, there is only one possible extension: the transformation is performed deterministically.

The formal definition from [23] is used:

Definition 6. Tiling is a function $f : \{0,1\}^* \to \{0,1\}^*$ defined as follows:

- If the input has the form (T, x) for a finite set of tiles T and a string x, and $x \xrightarrow{!}_T y$, then $f(T, x) = (T, y)$;[2]
- Otherwise, $f(T, x) = (T, x)$.

Using this definition, Kozhevnikov and Nikolenko prove the following theorem:

Theorem 1. If one-way functions exist, then Tiling is a weakly one-way function.

[2]Note that Levin as well as Kozhevnikov and Nikolenko choose to include the set T in both the input and output, but he could just as well have defined a tiling function as a function f whose domain and codomain are both the set of all finite sequences from a set T of tiles. In that case, we would describe the function's graph by writing $f(x) = y$, with no mention of T, as the set T would already be determined by the function's domain and codomain. However, here we choose to stick with Levin's original formulation.

This theorem and its proof are of great importance in the following sections of this paper, so we provide the proof along with an explanation of some details as well as a small, but necessary addition.

Proof. The proof revolves around proving that every polynomial time Turing machine can be simulated as a tiling problem. In fact, it suffices to restrict our attention to functions computable in quadratic time.[3] Let g be a length-preserving one-way function such that there exists a TM that computes g in no more than n^2 steps on inputs of length n.

Lemma 1. For every deterministic TM M with tape alphabet $\Gamma = \{0, 1, B\}$ working for no more than n^2 steps on inputs of length n, there exists a set of tiles T_M such that its set of labels $\mathcal{A}$ includes Γ, and $M(x) = y, |x| = |y|$, if and only if

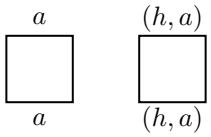
$$\$sxB^{n(n-1)}\# \xrightarrow{!}_{T_M} \$hyB^{n(n-1)}\#,$$

where s and h are the start and halt states of M, B is a blank symbol[4] and $\$, \# \notin \Gamma$ are special symbols marking the beginning and end of a string.

That is, at the start of the tiling process, the string $\$sxB^{n(n-1)}\#$ can be read from the bottom edges of the bottom row of tiles, and a full tiling of the square has the string $\$hyB^{n(n-1)}\#$ on the top row.[5] The proof of lemma 1 presents a construction of T_M, such that with this tileset the tiling grid can be properly filled for each computation on M.

Proof. The set of tiles T_M is constructed based on Turing machine M, by following these steps:

1. For each tape symbol $a \in \{0, 1, B\}$, we add



   This step adds tiles that serve as 'filler tiles'. Since they bring no change to the tiling (the symbols on top are the same as those on the bottom), they fill the grid with tiles on places where the head is not currently at. They can also be added to the tiling when the computation is finished but the square grid is not yet filled up until the top row.

[3] It has been proven by Levin that any weakly one-way function can be reduced to one that runs in quadratic time (see for example [27] or page 52 of [15]). Thus, the computation length may be bounded by any polynomial.

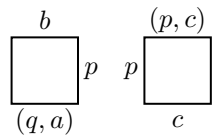
[4] The blank symbols can be replaced with symbols from $\{0, 1\}$ using a binary encoding. We will elaborate on this in the next chapter.

[5] In Levin's work from 2003, the input string is on the top of the grid. Kozhevnikov and Nikolenko define the transformation to be constructed upwards.

2. For each $a, b, c \in \{0, 1, B\}, q \in Q \backslash \{h\}, p \in Q$, if

$$\delta_M(q, a) = (p, b, R),$$

we add

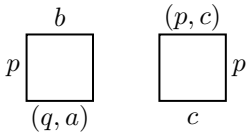


This step adds the tiles that simulate transitions with move-right commands. The first one simulates tape cells where the head has just moved right from, the second is for the adjacent tile (where the head ends up after moving right).

3. For each $a, b, c \in \{0, 1, B\}, q \in Q \backslash \{h\}, p \in Q$, if

$$\delta_M(q, a) = (p, b, L),$$

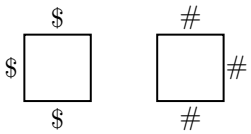
we add



Step 3 does roughly the same as step 2, but for transitions with move-left commands.

Remark. Unfortunately, there seems to be a small flaw in the proof from [23] covered here. Step 3 does not provide a way to simulate what happens when M carries out a move-left command while already on the leftmost cell. The correct behavior should be that the head remains on that cell, but we see that the presented tileset does not allow for this. It would be tempting to try to resolve this issue by simply adding an additional tile with (p, b) on the top edge and (q, a) on the bottom edge. However, this addition would result in further problems, such as the loss of uniqueness of a tiling. Therefore, for simplicity we leave the proof as presented in [23] intact here. Later, in the proof of Theorem 2, we show how to correct this.

4. For \$ and # we add



These tiles are put in the leftmost and rightmost column of the tiling grid, indicating respectively the left and right border of the tiling grid.

With this construction of the tileset, we see that each computation of M on input x that goes on for no more than n^2 steps[6], corresponds to a proper tiling of the $|x|^2 \times |x|^2$ square[7]. Namely, if the transition function provides a way to transform the input into the desired output string y, with $|x| = |y|$, it then follows that there exists a correct tiling of the grid, with the input at the bottom row and the output on the top row. □

Lemma 1 guarantees that there exists a finite set of tiles T_M such that

$$\$sxB^{n(n-1)}\# \xrightarrow{!}_{T_M} \$hyB^{n(n-1)}\#$$

is equivalent to $g(x) = y$. Therefore, with constant probability, inverting OWF g is equal to inverting Tiling. Tiling is therefore a complete weakly one-way function. By [23, Proposition 1], a strongly one-way function with the same property can be built. □

[6] See footnote 3.

[7] The change from an $|x| \times |x|$ grid to a $|x|^2 \times |x|^2$ one is caused by having input string x on the bottom row initially be followed by $|x| \cdot (|x| - 1)$ blank symbols for extra 'computation space'.

Chapter 3

Transfinite computation

3.1 Ordinal numbers

For a computation on a classic Turing machine, its steps are indexed by the natural numbers. The computational length is therefore bounded by ordinal number ω, which corresponds to the set of natural numbers $\mathbb{N}$. An algorithm might loop infinitely, resulting in a computation that will not cause the TM to halt. Thus TMs are not suited for supertask computations, i.e., computations that require infinitely many steps (see [30, 32]). It is clear that a machine different from the classic Turing machine is needed to perform such tasks: a machine that can perform more than ω many steps, by using ordinal numbers to go beyond the natural numbers. In order to understand how ordinal numbers can be applied this way, we present the basics of the theory of ordinals.

Ordinal numbers were first introduced in 1883 by Cantor, the founder of transfinite set theory. They are used to describe the order of a well-ordered set. A well-ordered set S is a set with a relation $<$ such that the following properties hold:

- **Trichotomy:** for any $x, y \in S$, exactly one of $x < y$ or $y < x$ or $x = y$ is true.
- **Transitivity:** for any $x, y, z \in S$, if $x < y$ and $y < z$, then $x < z$.
- **Well-foundedness:** every nonempty subset of S has a least element, which means that it has an element x such that there is no other element y in the subset for which $y < x$ holds.

Ordinal numbers are used to order the elements of a well-ordered set, and in this way they differ from cardinal numbers, which are used to quantify the number of elements in a set. Von Neumann defines the first few ordinals as follows [31]:

$$
\begin{array}{ll}
0 = \{\ \} & = \emptyset \\
1 = \{0\} & = \{\emptyset\} \\
2 = \{0,\ 1\} & = \{\emptyset, \{\emptyset\}\} \\
3 = \{0,\ 1,\ 2\} & = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} \\
4 = \{0,\ 1,\ 2,\ 3\} & = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}\} \\
... & ...
\end{array}
$$

This process is continued to construct all finite ordinals, which gives us a way to define the natural numbers in purely set-theoretic terms. After all the natural numbers comes the first infinite ordinal, denoted by ω. It is the ordinal corresponding to the set of natural numbers, and therefore the first ordinal that is bigger than any natural number.

Notice how each ordinal after 0 is the set of ordinals that precede it. That is, $n + 1 = n \cup \{n\}$, for all $n \in \omega$. For the finite ordinals, each ordinal has a maximum element. Ordinals that have a maximum element are successors of the ordinal corresponding to that maximum element, and are therefore called successor ordinals. Ordinal ω does not have a maximum element, since there is no largest natural number. An infinite ordinal with no maximum element, such as ω, is called a limit ordinal.

After the first limit ordinal ω, the list of ordinals continues with $\omega + 1$, $\omega + 2, ...$, which again are successor ordinals and are thus constructed in Von Neumann's manner. This goes on until $\omega + \omega$, also written as $\omega \cdot 2$, which is the second limit ordinal. The list then carries on with

$$\omega \cdot 2 + 1, \omega \cdot 2 + 2, ..., \omega \cdot 3, ..., \omega \cdot 4, ..., \omega \cdot \omega, ..., \omega^3, ..., \omega^4, ..., \omega^\omega, ..., \omega^{\omega^\omega}$$

and goes much further than that. For a more extensive overview of the growth of ordinals, see [29].

It is important to note that the ordinal arithmetic operations are non-commutative. For example, $\omega + 1$ is not equal to $1 + \omega$, which is in fact equal to ω. Non-commutativity also holds for multiplication and exponentiation. For more on ordinal arithmetic, refer to [21].

There are many models of computation that go beyond the Turing model. In the following sections we consider just two—the infinite time Turing machine and the ordinal Turing machine—with primary focus on the latter.

3.2 Infinite time Turing machines

In their 2000 paper, Hamkins and Lewis generalize the classic Turing machine to a machine that is allowed transfinite ordinal computation length [17]. It is called the infinite time Turing machine (ITTM). With the ITTM, an infinite amount of computation steps is allowed, providing a model for computing supertasks. The infinite time Turing machine model is not unlike that of the classic Turing machine. Computations are performed in similar manner, but there are several changes. In this paper we will not be working with the ITTM model, but with a different model that is based on it. It is therefore important to know how

the ITTM works, so we provide a brief description of how they differ from the classic TM.

- **ITTMs have three tapes.** These are an input tape, an output tape, and a scratch tape. The third is used for tasks such as storing intermediate results. In contrast to the case of finite TMs, which are computationally equivalent to multitape TMs [35, Theorem 3.13], the use of three tapes here is necessary: Hamkins and Seabold have proven that single-tape ITTMs are not fully equivalent to the three-tape models [18]. Like in classic TMs, the tapes have cells for every natural number.
- **Input may be of infinite length.** With ordinal computation time, a computation space of length ω and a binary alphabet, the input may be any element of Cantor space 2^ω. This way, computations on real numbers in $\mathbb{R}$ can be performed. Though the input tape of an ITTM is the same as that of a TM, a TM would be unable to read an input that is spread over the whole tape, since it does not have enough computation time to do so. Thus, an ITTM M computes a function $f : 2^\omega \to 2^\omega$ if for every input $x \in 2^\omega$, the machine halts with $f(x)$ on the tape (so $M(x) = f(x)$).
- **Special behavior at limit ordinal computation steps.** For computation steps indexed with successor ordinals, the ITTM behaves in the same way as a classic TM would. For limit ordinal times, the machine performs special limit behavior. Details are left out of the present paper, but are similar in many ways to the behavior of OTMs, which we see later. We refer to [17] for more detail.

ITTMs are more powerful than TMs. For example, they are capable of computing the halting problem, since they can simulate a finite TM and check whether ω many steps have passed. If the machine is still running, then the computation does not halt on regular TMs. However, ITTMs have their own halting problem they cannot decide. The transfinite version of the halting problem is deciding whether a given computation will halt after any transfinite ordinal number of steps. For ITTMs, Hamkins and Lewis identify two distinct and inequivalent versions of the halting problem:

$$H = \{<p, x> \mid \text{program } p \text{ halts on input } x\},$$

$$h = \{p \mid \text{program } p \text{ halts on input } 0\}.$$

H and h are both undecidable.

3.3 Ordinal Turing machines

ITTMs are more powerful than TMs, but they are limited in their power. Inspired by the findings of Hamkins and Lewis, Koepke introduced in 2005 a generalization of the ITTM, called the ordinal Turing machine [22]. Where the

ITTM extends the computational power of a classic TM by allowing computations of infinite ordinal length, Koepke saw the potential of extending the computation space, i.e., the tape length, to infinite ordinal length as well. The ordinal Turing machine (OTM) works somewhat differently from the ITTM.

- **The tape has one cell for every ordinal.** Instead of having one cell for every natural number, an OTM tape has one cell for every ordinal number. The tape of an OTM thus has transfinite ordinal length, which makes it possible for the head to move infinitely to the right, using the rules from below. If at any successor ordinal cell α the machine is instructed to move left, then the machine head will just move to cell $\alpha - 1$. However, if the index of the cell is any limit ordinal λ, the head cannot move to the left, since there exists no ordinal corresponding to $\lambda - 1$. The head position after a move-left command at *any* time t is therefore determined by

$$H(t+1) = \begin{cases} H(t) - 1, \text{ if } H(t) \text{ is a successor ordinal,} \\ 0, \text{ otherwise.} \end{cases}$$

  In other words, the head moves to cell 0 if it would be instructed to move left while at a limit ordinal cell.

- **Special behavior at limit ordinal times.** In OTMs, we calculate the tape content, the next state and the new head position at limit times with a limit inferior calculation over those previous values. In Koepke's OTM model, the possible values of the tape cells are taken over the tape alphabet $\Gamma = \{0, 1\}$. We adopt this convention in this work. At *limit* ordinal time t, i.e., after a limit ordinal number of computation steps, the content of each cell α on the tape T is determined by

$$T(t)_\alpha = \liminf_{s \to t} T(s)_\alpha.$$

  This thus updates the content of the whole tape using the *limit inferior* operation, which takes the least value for non-converging values. That is, if the value of a tape cell α switches cofinally often[8] between 0 and 1 as time s approaches t, the value does not converge (i.e., it does not take on either 0 or 1), so the least value, namely 0, is assigned at time t. Similarly, at that same time t the state S of the machine is determined by

$$S(t) = \liminf_{s \to t} S(s).$$

  This makes sure that if the machine state is looping cofinally often over a certain set of states, then the machine goes back to the first state in that set. Finally, the head position H at limit ordinal time t is determined by

$$H(t) = \liminf_{s \to t, S(s) = S(t)} H(s).$$

[8] We say that something occurs cofinally often before a (limit ordinal) time t if, for all $s < t$ where it occurs, there exists an s' where it occurs such that $s < s' < t$.

So, the head position at limit ordinal time t is determined by taking the lim inf value of the past head positions for which the machine was in the same state as it is at time t (which is $S(t)$).

- **Input may be of transfinite ordinal length.** Since an OTM has ordinal tape length, input of an OTM may have transfinite ordinal length. This increases the computational power of the OTM, compared with ITTMs [28, Theorem 3]. Thus, an OTM M computes a function $f : 2^\Omega \to 2^\Omega$ (where Ω denotes the set of all ordinals), if for every input $x \in 2^\Omega$, the machine outputs $f(x)$.

Koepke's definition of an OTM does not use blank tape symbols: both its input and tape alphabet generally are $\{0, 1\}$. For convenience, we could of course add a blank symbol in such a manner that it does not change the computational power of the model. This is done for the TM model in Kozhevnikov and Nikolenko's work [23].[9] We prefer leaving blank symbols out of the tape alphabet, since any addition of a symbol to the tape alphabet can be incorporated using binary encoding. For example, we can add a blank symbol by encoding 0 as 00, 1 as 11, and the blank by 01. This does not bring any changes to the computational power.

[9] If one chooses to add a new blank symbol B to the tape alphabet Γ, it is needed to specify how it behaves in the lim inf operation. Say for example that at some limit ordinal time t, the content of tape cell α is determined by $T(t)_\alpha = \liminf_{s \to t} T(s)_\alpha$, and the sequence of previous cell contents at cell α as time s approaches t is $B01B01B01...$. We define that in this case, value 0 is assigned at cell α at time t. Likewise, for sequence $B1B1B1...$, we assign value 1. This is a logical decision, since for non-converging values it is more natural to assign a value with meaning (i.e., 0 or 1) instead of a blank, since the cell does cofinally often contain information. Thus, for the lim inf operation, we define that blank symbol B counts as "more" than the other tape symbols 0 and 1.

Chapter 4

Transfinite one-way functions

Since Tiling is a complete one-way function, it provides the means to do further research on one-way functions. However, Tiling is constructed to simulate computations on classic Turing machines. It is of great interest to see if Tiling is complete in the context of transfinite models, such as ITTM and OTM, as well. This chapter shows that there exists a transfinite complete one-way function, when provided with a suitable system of generalized probability.

4.1 Tiling Expansion for OTMs

In order to prove that there exists a complete one-way function for ITTMs or OTMs, we will show that Tiling (as it is defined by Kozhevnikov and Nikolenko) can be generalized to OTMs, and that this generalization is complete weakly one-way as well. We begin by generalizing Levin's $n^2 \times n^2$ tiling grid to an $\alpha^2 \times \alpha^2$ grid, where α is a (possibly transfinite) ordinal. Since a computation on an OTM is 'shaped' like a square (it has ordinal α tape space and α time), and an ITTM computation has rectangular shape (ω tape space and α time), it is more natural to make use of the OTM model when generalizing Tiling to the transfinitary context.

4.2 A probabilistic OTM

Definitions 2 and 3 make use of probabilistic Turing machines. To see if one-way functions are possible for ordinal Turing machines, it is therefore necessary to define a probabilistic OTM. The challenge that occurs is that the fraction $\frac{1}{p(\alpha)}$ in these definitions becomes infinitely small, if α is an infinite ordinal. These infinitely small values are called *infinitesimals*, and are not included in the set of

real numbers. To answer this challenge, we propose the use of non-Archimedean probability as a potential answer in the following section.

A related problem is that we must get clear on how to use the notion of a uniform probability distribution in the transfinitary context. In the finitary case, it is straightforward to consider the set of all strings of a given length n (this set has 2^n elements) and thus find that the probability of a randomly chosen string being a particular given string is $\frac{1}{2^n}$. In the infinitary case, things are more complex. It is well known that a uniform distribution over a countably infinite set is not possible [20]. However, in our situation we never need to do that, because the cardinal 2^α is uncountable for all infinite α. Still, some work is needed to get clear on how to define uniform probability distributions for our purposes. It is clear that further study is needed to identify a suitable generalized notion of probability.

4.2.1 Non-Archimedean probability

To be able to work with infinitesimal probability, Benci *et al.* proposed the notion of non-Archimedean probability (NAP) [1]. It alters the concept of classical probability, which is formalized by the axioms of Kolmogorov (K0-K4).

The following formulation of Kolmogorov's axioms is from [3].

Definition 7. A triple $\langle \Omega, \mathfrak{A}, P_K \rangle$, where Ω is the sample space, $\mathfrak{A}$ is the event space, and P_K is the probability function, is called a *classical probability space* if the following axioms hold:

K0. **Domain and Range:** The events are the elements of $\mathfrak{A} \subseteq \mathcal{P}(\Omega)$ (where $\mathcal{P}(\Omega)$ denotes the powerset of Ω) and the probability function is a function

$$P_K : \mathfrak{A} \to \mathbb{R}.$$

K1. **Non-negativity:** $\forall A \in \mathfrak{A}$,

$$P_K(A) \geq 0.$$

K2. **Normalization:**

$$P_K(\Omega) = 1.$$

K3. **Additivity:** If A and B are events and $A \cap B = \emptyset$, then

$$P_K(A \cup B) = P_K(A) + P_K(B).$$

K4. **Continuity:** Let

$$A = \bigcup_{n \in \mathbb{N}} A_n,$$

where $A_n \subseteq A_{n+1}$ are elements of $\mathfrak{A}$; then

$$P_K(A) = \lim_{n \to \infty} P_K(A_n).$$

This combination of axioms rules out the possibility of infinitesimals [1, Remark 1]. NAP is proposed as a solution. It uses axioms similar to the Kolmogorov axioms, but most notably replaces the continuity axiom.

Again citing from [3], the first four axioms of NAP are:

NAP0. **Domain and Range:** The events are all the subsets of Ω, which is a finite or infinite sample space. Probability is a total function

$$P : \mathcal{P}(\Omega) \to \mathfrak{R},$$

where $\mathfrak{R}$ is a superreal field (that is, an ordered field that contains the real numbers as a subfield).

NAP1. **Regularity:** $P(\emptyset) = 0$ and $\forall A \in \mathcal{P}(\Omega)\backslash\{\emptyset\}$,

$$P(A) > 0.$$

NAP2. **Normalization:**

$$P(\Omega) = 1$$

NAP3. **Additivity:** If A and B are events and $A \cap B = \emptyset$, then

$$P(A \cup B) = P(A) + P(B).$$

These four axioms are not enough to give a full notion of NAP, since with the usual Archimedean limit operation $\lim_{n\to\infty}$, it is impossible to calculate an infinite sum of disjunctive events. K4 should therefore be replaced with an axiom that does allow this. The answer lies with the following principle:

Conditional Probability Principle (CPP): Let $\{\lambda_n\}$ be a family of events such that $\lambda_n \subseteq \lambda_{n+1}$ and $\Omega = \bigcup_{n\in\mathbb{N}} \lambda_n$; then, eventually

$$P_K(\lambda_n) > 0,$$

and, for any event A, we have that

$$P_K(A) = \lim_{n\to\infty} P_K(A|\lambda_n).$$

CPP is proven to be equivalent to K4 [1, Theorem 4], and is used to form the fifth and final axiom of non-Archimedean probability:

NAP4. **Conditional Probability Principle (CPP) in NAP:** For any nonempty $A \in \mathcal{P}(\Omega)$ and any $\lambda \in \mathcal{P}_{fin}(\Omega)$, where $\mathcal{P}_{fin}(\Omega)$ is the collection of *finite* subsets of Ω,

$$P(A|\lambda) \in \mathbb{R}$$

and

$$P(A) = \lim_{\lambda\uparrow\Omega} P(A|\lambda),$$

where $\lambda \uparrow \Omega$ denotes the 'Ω–limit' operation, which is a special kind of limit, different from the usual Archimedean limit $\lim_{n\to\infty}$. It essentially calculates the value at infinity of a special function defined for any λ. Further details can be found in [1].

By replacing K4 with NAP4, infinite sums of disjunctive events are allowed. This way, NAP is applicable to infinite domains. This notion of probability as well as the existence of infinitesimal probabilities are subject to some criticism on philosophical grounds. Benci *et al.* have defended against some of this criticism [3]. Though their NAP model is mainly a proposed model, it is a promising candidate for use in our present purposes.

The NAP model allows the use of infinitesimals, but these are not real numbers. The *surreal numbers* are a superset of the real numbers, containing infinitesimal numbers. For more on surreal numbers, including their construction and arithmetic, see [10].

Though more research is needed to settle on a generalization of probability theory suitable for defining probabilistic OTMs, for the rest of this paper we leave this issue aside and take the definition of a probabilistic OTM as a given. Thus we can now define, for all such suitable generalizations, what it means for a function to be weakly transfinite one-way.

Definition 8. A function $f : \{0,1\}^* \to \{0,1\}^*$ is called *weakly transfinite one-way* if the following conditions hold:

1. There exists a deterministic ordinal Turing machine M with input alphabet $\{0,1\}$ that computes f in ordinal polynomial time;

2. There exists an ordinal polynomial p such that for every probabilistic ordinal Turing machine M' and for every sufficiently large ordinal input length α,

$$Pr_{M',x\in U_\alpha}[M'(f(x),1^\alpha) \notin f^{-1}(f(x))] > \frac{1}{p(\alpha)},$$

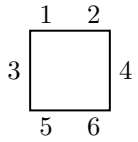
where the probability is taken over the random bits of M' and input x (both distributions are uniform).

This definition uses ordinal polynomial time. Just as we have polynomial time for finite numbers, there is also such a notion for ordinals, in which the variable(s) in the polynomial may be any ordinal number. In the transfinitary context, we use ordinal polynomials in place of finitary polynomials. As was mentioned in Section 3.1, ordinal arithmetic differs from regular arithmetic since addition, multiplication and exponentiation are not commutative. In our case we generally use polynomials that have an ordinal variable, i.e., an input of ordinal length α, raised to the power of a natural number k. However, the exponent could also be an ordinal β. For more on ordinal polynomial time, we refer to [6], which introduces the theory of computational complexity for OTMs.

4.3 A complete one-way function for OTMs

Now that a definition of a weakly transfinite one-way function has been given, a generalization of Tiling into one such function is proposed next. The construction of the proof that Tiling is complete one-way from Kozhevnikov and Nikolenko [23] is closely followed. The definition of the new *transfinite tiling simulating function* (or Transfinite Tiling) is equivalent to Definition 6. It does not change when generalized to the transfinitary context, since the transition function δ of an OTM is finite. The set of tiles T therefore remains finite as well.

The generalization of Tiling into Transfinite Tiling starts with redefining the concept of a *tiling process* into a *transfinite tiling process*, as follows. In a transfinite tiling process, a *tile* is a square whose edges are marked with symbols from a finite set $\mathcal{A}$ of symbols, and the positions where symbols may be placed are indexed as follows:



We assume a transfinite supply of tiles of each kind. In the above, we specify that positions 1 and 2 are only required to match respectively positions 5 and 6 of the tile above it. They do not have to match any symbols on the side edges of other tiles. Thus, they are not 'corner labels', but only top- and bottom edge labels. Furthermore, the symbol on position 3 of a given tile should match the symbol on position 4 of the tile left from it. Depending on a tile's purpose, some symbol spots may be empty, though empty spots have to match to adjacent empty spots as well.

This transfinite version of a tile differs from the finite version in that the symbols on the top and bottom edges are separated into two symbols. This conveniences computing the values of the symbols during the tiling process, which will become clear in the definition below. For sake of symmetry, we could have designed the tile to have two symbols on the left and right edge as well, totaling to 8 symbols per tile. However, one symbol on both the left and right edges suffices for our proof.

Naturally, it makes sense to maintain the same notions of matching edges at successor ordinal rows and columns, but we also have to specify what is done at limit ordinal rows and columns. For these cases, we use the limit inferior operation. Additionally, we specify that the set of symbols that are possible at a certain position is ordered, where an empty spot counts as "more" than any other symbol value that is involved in the lim inf operation. These requirements become clear in the following definition:

Definition 9. A *lim-inf transfinite tiling* of an $\alpha \times \alpha$ square is a set of α^2 tiles covering an $\alpha \times \alpha$ grid such that:

1. The symbols on adjacent edges of tiles match, as described above.

2. At every limit-ordinal numbered row λ, for every column τ, the position 6 symbol of tile (τ, λ) in that row is s, where

$$s = \liminf_{\rho \to \lambda} \{t \mid t \text{ is the position 2 symbol of tile } (\tau, \rho)\}.$$

3. For every row γ, let x_γ denote the least position 5 symbol in tiles from row γ. Then, at every limit ordinal numbered row λ, the following hold:

    1. The symbol x_λ is determined as $x_\lambda = \liminf_{\rho \to \lambda} x_\rho$.

    2. The symbol x_λ appears in position 5 of tile (θ, λ), where

$$\theta = \liminf_{\rho \to \lambda} \{\mu \mid x_\lambda \text{ occurs in position 1 of tile } (\mu, \rho)\}.$$

    3. In tiles (β, λ), where $\beta \neq \theta$, the symbol of position 5 is y, where

$$y = \limsup_{\rho \to \lambda} \{z \mid z \text{ is the position 1 symbol of tile } (\beta, \rho)\}.$$

4. At every limit-ordinal numbered column χ, the symbol on position 3 of tiles (χ, σ) in that column is v, where

$$v = \liminf_{\xi \to \chi} \{w \mid w \text{ is the position 4 symbol of tile } (\xi, \sigma)\}.$$

The goal of this definition is to describe a function that we can prove is transfinite complete one-way. We admit that this definition of a lim-inf transfinite tiling does in some ways import features of OTMs, but it is not entirely contrived. Every transfinite Tiling definition should implement some rule for the special behavior at limit indexed cells, and we have tried doing so in a way that it is not completely ad hoc. We will get back to this in chapter 5, but for now we continue with our proof.

Theorem 2. If one-way functions exist for OTMs, then Transfinite Tiling is a weakly one-way function for OTMs.

Proof. Let g be a length-preserving one-way function for OTMs, such that there exists an OTM that computes g and works for no more than α^ζ steps, where ζ is any ordinal,[10] on inputs of length α.

It will now be proven that every OTM can be modeled as a transfinite tiling problem. That is, every computation on an OTM with input of length α corresponds to a lim-inf transfinite tiling of an $\alpha^\zeta \times \alpha^\zeta$ square.

[10]It was out of the scope of this paper to prove that the findings of footnote 3 hold for transfinite weakly one-way functions as well. We therefore use computations bounded by α^ζ steps instead.

Lemma 2. For every deterministic ordinal Turing machine M' working for no more than α^ζ steps on inputs of length α, there exists a set of tiles $T_{M'}$ such that $M'(x) = y$ with $|x| = |y|$, if and only if

$$\$sx'B^{\alpha^\zeta} \xrightarrow{!}_{T_{M'}} \$hy'B^{\alpha^\zeta},$$

where:

- s is the start state of M',
- h is its halting state,
- B is a blank symbol that is encoded using symbols from $\{0, 1\}$,
- $\$ \notin \Gamma$ is an additional symbol marking the beginning of a string,[11]
- x' (respectively y') is input string x (respectively output string y), modified such that its first symbol $r \in \{0, 1\}$ is replaced by special symbol r_0 (i.e., the same symbol r, but with a subscript 0), and every occurrence of a symbol $t \in \{0, 1\}$ in a limit ordinal position within the strings, is replaced by a new special symbol t_l (i.e., the same symbol t, but with a subscript l).

Note that the input and output strings are followed by α^ζ many blank symbols, as opposed to $n(n-1)$ many in [23] (see also the proof of Theorem 1 of the current paper). This is because we are working with ordinals, and there exists no value $\alpha - 1$ if α is a limit ordinal. However, ordinal arithmetic teaches us that $\alpha + \alpha^\zeta = \alpha^\zeta$ [8, 21], so we may construct the tiling square to have α^ζ many blank symbols.

Proof. Consider an ordinal Turing machine

$$M = (Q, \{0, 1\}, \{0, 1\}, \delta, s, h).$$

For the lim-inf transfinite tiling, we define the set of symbols $\mathcal{A}$ as

$$\mathcal{A} = \{a, b, c, a_0, b_0, a_l, b_l, @_q, s, h, \%, \$\},$$

where:

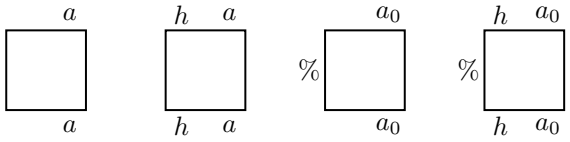
- a, b, c are symbols for denoting elements of the tape alphabet,
- a_0, b_0 are symbols denoting the elements in the tape alphabet when they appear at the start of the string,
- a_l, b_l are symbols denoting the elements in the tape alphabet when they appear at a limit ordinal index of the string,

[11] The special symbol # that was used in Kozhevnikov and Nikolenko's work [23] is excluded from this definition. This is because, if we have infinite ordinal input length, there exists no real 'end' of the input string. It therefore is not natural to bind the tiling grid using a special symbol.

- $@_q$ are special symbols for each $q \in Q \backslash \{h\}$,
- s, h are the start and halting states,
- % is a special symbol for connecting the leftmost column to the column corresponding to the first index of the string,
- \$ is a special symbol for denoting the leftmost column, and thus the left end of the tape.

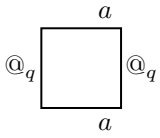
Using $\mathcal{A}$, we follow these steps to construct the tileset T_M:

1. For special symbol %, for halting state h and for each $a, a_0 \in \{0, 1\}$, we add:



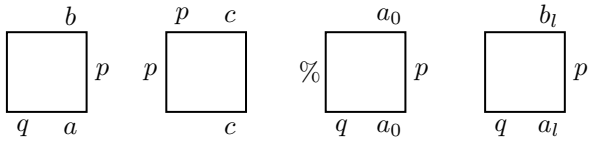
   The first two tiles of this step are the transfinite tiling version of the filler tiles from step 1 of the proof of lemma 1. The third and fourth tile are similar tiles for the column for the first input symbol, which connects to the leftmost column by the % symbol. The subscript 0 on the symbols in these tiles serves to distinguish this column from all other columns. The need for this distinction becomes clear in step 4.

2. For each special symbol $@_q$ such that $q \in Q \backslash \{h\}$, and for each $a \in \{0, 1\}$, we add:



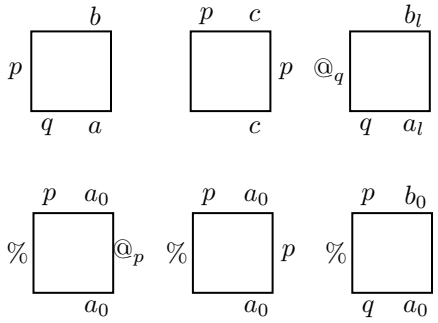
   This step adds a tile for each pair in $Q \backslash \{h\} \times \{0, 1\}$, with a special symbol $@_q$. This way, a row of only these tiles can be used to simulate moving the head back to the leftmost cell when a move-left command is simulated in a limit ordinal indexed column. See also step 4 below.

3. For each $a, b, c \in \{0, 1\}$, $q \in Q \backslash \{h\}$, $p \in Q$, if $\delta(q, a) = (p, b, R)$, we add:



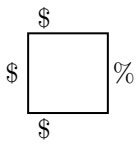
This step adds the tiles that simulate transitions with move-right commands. The first two are transfinite tiling versions of the tiles from step 2 of the proof of lemma 1. The third and fourth play the same role as the first tile, but for use in the zero and limit ordinal indexed columns respectively.

4. For each special symbol $@_q$ such that $q \in Q \backslash \{h\}$, for each $p \in Q$ and for each $a_0, a_l, b_l, a, b, c \in \{0, 1\}$, if $\delta(q, a) = (p, b, L)$, we add:



Step 4 does roughly the same as step 3, but for transitions with move-left commands. Different is the third tile, which is for moving left in limit ordinal columns. This tile forces all the tiles to its left in that row to be tiles from step 2, except for the tile in the column corresponding to the first index of the string, which is forced to become the fourth tile type from step 4. This shows why we need to distinguish the first tape symbol from the others, using the subscript 0. The fifth tile is used for moving to the column corresponding to the first index of the string, from the adjacent tile on its right. The sixth tile is used for simulating transitions with move-left commands when the head is at the leftmost cell. There, the simulated head remains in the same column when a move-left command is simulated.

5. For special symbols \$ and %, we add:



These tiles are put in the leftmost column of the tiling grid, indicating the left border of the tiling grid. Using special symbol %, this makes sure that tiles with a_0 and b_0 symbols can only be placed in the adjacent column, since other tiles do not have a % symbol on their left edge.

Using this construction of the tileset, each computation of M corresponds to a proper tiling of the square. We can now see that for each computation on the deterministic ordinal Turing machine M on input x (where x may be of any ordinal length α), that is bounded by α^ζ steps, there exists a corresponding tiling of the $\alpha^\zeta \times \alpha^\zeta$ square in which bottom labels of the lowest row form the input string (padded with symbols B to length α^ζ), and top labels of the uppermost row form the output string (padded with symbols B to length α^ζ). If the computation halts after fewer than α^ζ steps, we can still finish the tiling with tiles from step 1. □

Lemma 2 guarantees that there exists a finite set of tiles T_M such that

$$\$sx'B^{\alpha^\zeta} \xrightarrow{!}_{T_M} \$hy'B^{\alpha^\zeta}$$

is equivalent to $g(x) = y$. Therefore, with constant probability[12] (equal to the probability of the event that the input string begins with the description of T_M), inverting Transfinite Tiling is equivalent to inverting g. □

Thus, it has been proven that Transfinite Tiling is a transfinite complete weakly one-way function for ordinal Turing machines, relative to a suitable system of generalized probability. From [23, Proposition 1], we know that a strongly one-way function with the same property can be built from a weakly one-way function.

[12]This is a classical, finite probability, so it does not depend on the notion of transfinite probability used in the definition of probabilistic OTMs.

Chapter 5

Conclusion and discussion

The aim of this paper was to formulate a notion of a one-way function for transfinite computation and to answer the following research question: *Does a transfinite complete one-way function exist?* The work started by motivating the research. First, we gave a summary of the classic Turing machine model as well as a brief explanation of the infinite time Turing machine [17], followed by a description of the ordinal Turing machine [22]. Next, we gave an introduction to one-way functions and formulated a definition of a transfinite one-way function.

We constructed an answer to the research question by answering two subquestions. The first was: *Can a probabilistic ordinal Turing machine be defined?* It was outside the scope of this paper to provide a concrete answer to this question. For a notion of probability on infinite sets we presented non-Archimedean probability, which has been proposed by Benci *et al.* [3]. Further research is needed in order to conclude whether this notion indeed suits the OTM model, as well as to determine what number domain should be used to include infinitesimals in the probability model. For the latter problem, we suggested the use of surreal numbers.

The second subquestion was: *Can Tiling be generalized in such a manner that it becomes a transfinite complete one-way function that simulates ordinal Turing machines?* By carefully altering the definition of a tiling of an $n^2 \times n^2$ square [23] into a lim-inf transfinite tiling of an $\alpha^\zeta \times \alpha^\zeta$ square, as well as its completeness proof, where the construction of the tiling set was changed, we presented a transfinite complete weakly one-way function that simulates an ordinal Turing machine. It has been named Transfinite Tiling.

These results tell us that it is indeed possible to come up with a transfinite complete one-way function, given any suitably generalized theory of probability. Further research may be directed toward finding the most appropriate such theory. In the present paper we have suggested the use of surreal numbers and non-Archimedean probability, or a similar non-classical theory of probability. An alternative approach would use another way of measuring the relative "size" of a set of outcomes in a probability space to the whole probability space, for instance via numerosities [2] or the concept of meager sets.

In defining the notion of transfinite tiling in this paper, it was mentioned that some readers might find the definition a bit ad hoc, since it arguably imports some features from the definition of an OTM. Nevertheless, it works for our present purposes of defining a transfinite complete one-way function. Seeing as the original aim of Levin's 2003 paper on tiling was to find a more 'natural' complete one-way function than what he found in 1987, it would likewise be good for us in future research to find a more natural definition of transfinite tiling that depends less on the specific functioning of OTMs.

The findings of this work contribute to the research field of transfinite computation. They provide insight into the application of one-way functions for OTMs, which points to potential further research. For example, it might be interesting to see if it is possible to design a similar construction of a transfinite complete one-way function for ITTMs. Additionally, one could try to define a notion of a transfinite pseudorandom number generator in terms of OTMs, as they have been proven to be equivalent to one-way functions in the finitary context [19]. Like one-way functions, pseudorandom number generators are of great use in the field of cryptography, and it is undoubtedly of interest to research what other possibilities occur with the use of transfinite computational models. The field of transfinite cryptography [33] is a young and as of yet not deeply explored subfield, but it is reasonable to expect that transfinite one-way functions will be a relevant piece of the puzzle.

Bibliography

[1] Benci, V., Horsten, L., Wenmackers, S. (2013). Non-Archimedean Probability. *Milan Journal of Mathematics*, 81(1), 121–151. https://doi.org/10.1007/s00032-012-0191-x

[2] Benci, V., Bottazzi, E., Di Nasso, M. (2014) Elementary numerosity and measures. *Journal of Logic and Analysis,* 1-14. https://doi.org/10.4115/jla.2014.6.3

[3] Benci, V., Horsten, L., Wenmackers, S. (2018) Infinitesimal Probabilities. *British Journal for the Philosophy of Science*, 69(2), 509-552. https://doi.org/10.1093/bjps/axw013

[4] Berger, R. (1966) Undecidability of the Domino Problem. *Memoirs of the American Mathematical Society*, 66. https://doi.org/10.1090/memo/0066

[5] Bianchetti, M. (2020) Weaker variants of infinite time Turing machines. *Archive for Mathematical Logic*, 59, 335–365. https://doi.org/10.1007/s00153-019-00692-9

[6] Carl, M., Löwe, B., Rin, B.G. (2017) Koepke Machines and Satisfiability for Infinitary Propositional Languages. *Lecture Notes in Computer Science*, 10307, 187-197. https://doi.org/10.1007/978-3-319-58741-7_19

[7] Cantor, G. (2005) Paper on the 'Foundations of a General Set Theory'. *Landmark Writings in Western Mathematics 1640-1940, Elsevier Science*, 46, 600-612. https://doi.org/10.1016/B978-0-444-50871-3.X5080-3

[8] Clarck, J. R. (2017) Transfinite Ordinal Arithmetic. Master's thesis at Governors State University.

[9] Cobham, A. (1965) The Intrisic Computational Difficulty of Functions. *Logic, Methodology and Philosophy of Science: Proceedings of the 1964 International Congress.* Ed. Yehoshua Bar-Hillel. *North-Holland Publishing,* 24-30. https://doi.org/10.2307/2270886

[10] Conway, J. H. (2000) On Numbers and Games, 2nd edition. *CRC Press.*

[11] Cook, S. A. (1971) The Complexity of Theorem-Proving Procedures. *Proceedings of the Third Annual ACM Symposium on the Theory of Computing*, 151-158. https://doi.org/10.1145/800157.805047

[12] Deolalikar, V., Hamkins, J.D., Schindler, R. (2005). $P \neq NP \cap$ co-NP for Infinite Time Turing Machines. *Journal of Logic and Computation*, 15(5), 577–592. https://doi.org/10.1093/logcom/exi022

[13] Dik, J. F. (2018). Infinite Time Turing Machines and Ordinal Turing Machines. Comparison and analysis of the computational power. Bachelor's thesis at University of Amsterdam. Available from supervisor.

[14] Gasarch, W. (2002). The P=?NP poll. *SIGACT News*, 33(2), 34-47.

[15] Goldreich, O. (2001). Foundations of Cryptography. Basic Tools. *Cambridge University Press.*

[16] Goldreich, O. (2008). Computational Complexity: a Conceptual Perspective. *Cambridge University Press.*

[17] Hamkins, J. D., Lewis, A. (2000). Infinite Time Turing Machines. *Minds and Machines*, 12(4), 521-539. https://doi.org/10.1023/A:1021180801870

[18] Hamkins, J.D., Seabold, D.E. (2001). Infinite time Turing machines with only one tape. *Mathematical Logic Quarterly*, 47(2), 271-287. https://doi.org/10.1002/1521-3870(200105)47:2¡271::AID-MALQ271¿3.0.CO;2-6

[19] Håstad, J., Impagliazzo, R., Levin, L. A., Luby, M. (1999). A Pseudorandom Generator from any One-Way Function. *SIAM Journal on Computing*, 28(4), 1364–1396. https://doi.org/10.1137/S0097539793244708

[20] Howson, C. (2013). Finite additivity, another lottery paradox and conditionalisation. *Synthese*, 191(5), 989–1012. https://doi.org/10.1007/s11229-013-0303-3

[21] Jech, T. (2007). Set Theory. *Springer Publishing.*

[22] Koepke, P. (2005). Turing Computations on Ordinals. *The Bulletin of Symbolic Logic*, 11(3), 377-397. https://doi.org/10.2178/bsl/1122038993

[23] Kozhevnikov, A. A., Nikolenko, S. I. (2009). On Complete One-Way Functions. *Problems of Information Transmission*, 45(2), 168-183. https://doi.org/10.1134/S0032946009020082

[24] Levin, L. A. (1973). Universal Sequential Search Problems. *Problemy Peredachi Informatsii 9*, 3, 115-116.

[25] Levin, L. A. (1987). One-Way Functions and Pseudorandom Generators. *Combinatorica*, 4, 357-363. https://doi.org/10.1007/BF02579323

[26] Levin, L. A. (2003). The Tale of One-Way Functions. *Problems of Information Transmission*, 39, 92-103. https://doi.org/10.1023/A:1023634616182

[27] Liu, M. (2013). Reductions for One-Way Functions. Bachelor's thesis at The University of Michigan.

[28] Löwe, B. (2006) Space Bounds for Infinitary Computation. *Lecture Notes in Computer Science*, 3988, 319-329. https://doi.org/10.1007/11780342_34

[29] Madore, D. A. (2017). A Zoo of Ordinals. Unpublished manuscript.

[30] Manchak, JB, Roberts, B.W. Supertasks. (2016) *The Stanford Encyclopedia of Philosophy* (Winter 2016 Edition), Edward N. Zalta (ed.). https://plato.stanford.edu/archives/win2016/entries/spacetime-supertasks/

[31] Von Neumann, J. (2009) Zur Einführung der transfiniten Zahlen. *Acta Sci. Math. (Szeged)* 1:4-4(1922-23), 199-208 6165/2009.

[32] Ord, T. (2002). Hypercomputation: Computing More than the Turing Machine. *CoRR*, Department of Computer Science, University of Melbourne.

[33] Patarin, J. (2012). Transfinite Cryptography. *International Journal of Unconventional Computing*, 8(1), 61-72.

[34] Rin, B. G. (2014). The computational strengths of α-tape infinite time Turing machines. *Annals of Pure and Applied Logic*, 165(9), 1501-1511. https://doi.org/10.1016/j.apal.2014.04.016

[35] Sipser, M. (2006). Introduction to the Theory of Computation. *Thomson Course Technology.*

[36] Turing, A. M. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1), 230–265. https://doi.org/10.1112/plms/s2-42.1.230