Space Complexity in Infinite Time Turing Machines

MSc Thesis (*Afstudeerscriptie*)

written by

Joost Winter
(born 20th of March, 1980 in Amsterdam, the Netherlands)

under the supervision of **Benedikt Löwe**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
14th of August, 2007	Peter van Emde Boas
	Joel David Hamkins
	Benedikt Löwe
	Jouko Väänänen

Institute for Logic, Language and Computation

Contents

Chapter 1

Introduction

This Master's thesis will deal with the concept of infinite time Turing machines, as originally defined in [HaLe]. Infinite time Turing machines are defined in a way similar to ordinary Turing machines, but can continue to operate at transfinite ordinal stages. Although defined similarly, however, there are some striking differences between some of the established results about infinite time Turing machines, and the corresponding results for regular Turing machines. For example, it has been shown in [Sc] that the infinite time analogue of the problem $\mathbf{P} = \mathbf{NP}$ is false, and it was proven in [HaSe] that infinite time Turing machines with only one tape are less powerful than machines with n tapes for any $n > 1$.

As it turned out that an analogue of a problem that is known to be extremely difficult in the classical case—the question whether $\mathbf{P} = \mathbf{NP}$—is quite simple to solve in the case of infinite time Turing machines, we may start to wonder if we can do similar things for analogues of other problems, that are also known to be 'hard' in the case of classical Turing computation. One such problem is the question whether $\mathbf{P} \subsetneq \mathbf{PSPACE}$, which will be one of the main questions dealt with in this thesis. Another question we will consider—one which *is* solved in the classical case—is the question whether, in the case of infinite time Turing machines, the equality $\mathbf{PSPACE} = \mathbf{NPSPACE}$ holds.

More precisely, in this thesis we will provide definitions for the classes $\mathbf{PSPACE}_f$ and $\mathbf{NPSPACE}_f$ for functions f from sets of natural numbers to ordinals, following the definitions in [Lö], and attempt to tackle a number of open questions related to these classes, such as:

- For which functions f do we have $\mathbf{P}_f = \mathbf{PSPACE}_f$?
- For which functions f do we have $\mathbf{PSPACE}_f = \mathbf{NPSPACE}_f$?

Following this, we will also consider another model for infinite Turing computation, as provided by Peter Koepke, in which we do not only have computations of infinite length, but also tapes which have the size of the class of all ordinals. Here we will again consider similar questions about space complexity, and also establish some relationships between these classes and the complexity classes of the standard model.

1.1 Overview

In the second chapter of this thesis, we will restate the standard definitions of the model of transfinite Turing computation that was originally defined by Joel Hamkins and Jeffrey Kidder. Several variations on this model, involving varying numbers of tapes, additional scratch cells, and modified rules for limit ordinals, will be considered and, where possible, the equivalence of the computational powers of these models will be proven.

In the third chapter, the notions of clockability and writability of ordinals will be considered, and a number of important results will be restated.

In chapter 4, we will state the definitions of the space and time complexity classes $\mathbf{P}_f$ and $\mathbf{PSPACE}_f$, as well as of the weak halting problem h. Following this, we will show the invariance of some of these classes under the different models of infinite time Turing machines, and under addition of extra tapes and/or scratch cells.

In chapter 5, we will consider the question $\mathbf{P}_f \stackrel{?}{=} \mathbf{PSPACE}_f$ for a variety of functions f: it will turn out that, for some functions f, $\mathbf{P}_f$ is a strict subset of $\mathbf{PSPACE}_f$, and for other functions, the two classes are equal.

Following this, in chapter 6, we will consider Koepke's model of transfinite Turing computation; give definitions of the space and time complexity classes; and state and prove a number of results.

In chapter 7, we will define the indeterministic space and time complexity classes **NP** and **NPSPACE**, and consider the relationship between the classes $\mathbf{PSPACE}_f$ and $\mathbf{NPSPACE}_f$, as well as between the classes $\mathbf{NP}_f$ and $\mathbf{NPSPACE}_f$, for a variety of functions f, in the case of both the Hamkins-Kidder model and the Koepke model.

Finally, in chapter 8 some general conclusions, remarks, and further ideas will be provided.

I would like to thank to Benedikt Löwe, my thesis supervisor, Joel David Hamkins and Philip Welch, as well as to the people present at the Bonn International Workshop on Ordinal Computability in January 2007 for help, advice, and/or critical reading.

1.2 Notation

In this thesis, the set $\mathbb{N}$ of natural numbers will always start with 0, and never with 1. The symbol $\mathbb{R}$ will always refer to $\mathcal{P}(\mathbb{N})$, as usual in e.g. descriptive set theory and all earlier literature on infinite time Turing machines, and the term 'real' will always refer to a subset of $\mathbb{N}$.

At some points, we will consider natural numbers n as reals: here we consider a natural number n as equal to the set $\{0, \ldots, n-1\}$, which clearly is a real according to our definitions. We also will, on several occasions, identify the natural numbers with the finite ordinals: the set $\mathbb{N}$ and the ordinal ω refer to the same object.

The classes **P**, **NP**, **PSPACE**, etc. will always refer to the infinite time variants of these problems: no illusions about any breakthrough in the usual $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ problem should be entertained. Other than this, the notations in this thesis will be in accord with the standard notations.

Whenever we say that something happens 'cofinally often' before a certain ordinal α, this means that for every ordinal $\beta < \alpha$, there is an ordinal γ with $\beta < \gamma < \alpha$, such that the event happens at γ.

Besides the usual and familiar notions of ordinal addition and multiplication, in a few cases we will also use the less common notion of ordinal subtraction:

Definition 1.1. If α and β are ordinals, we say that $\alpha - \beta = 0$ if $\alpha < \beta$, and if $\alpha \geq \beta$, then $\alpha - \beta$ denotes the unique ordinal η, such that $\beta + \eta = \alpha$.

Notions from set theory, in particular ordinal arithmetic, definitions by ordinal recursion, and proofs by ordinal induction, will be used quite freely in this thesis.

We will use the letters a, b, d, e for finite sequences of 0s and 1s; A, B, C for arbitrary sets; i, k, m, n for natural numbers; v, w, x, y for reals; f, g for functions; S, T for Turing machines; and $\alpha, \beta, \eta, \theta$ for ordinals.

Chapter 2

Infinite time Turing machines

In this chapter, we will start by defining the standard model of infinite time Turing machines, which is essentially the same as the model described in [HaLe]. Following that, we will consider a number of variations on this model, differing from the standard model in various ways: for instance, we will define machines with a different number of tapes, machines with a different set of symbols that could be written on the tape, and machines with a different type of operation, and discuss a number of equivalence results.

2.1 The standard model, ITTM_0

The hardware of an infinite time Turing machine will be mostly identical to that of a regular Turing machine, with the main difference that, in the standard configuration, an infinite time Turing machine uses three tapes instead of one—an input tape, an output tape, and a scratch tape. The computation, on the other hand, differs in the important sense that we allow an infinite time Turing machine to continue and finish, in principle, computations at any ordinal stage, by means of special rules for the computation at limit ordinals. Definition 2.1 below will be effectively equal to, but more precise than, the one given in [HaLe].

Definition 2.1. An *infinite time Turing machine*, a *Hamkins-Kidder machine*, or an ITTM_0 *machine* is defined by a tuple of the form

$$T = (Q, \delta, q_{\mathrm{s}}, q_{\mathrm{lim}}, q_{\mathrm{f}})$$

where

- Q is a finite set of internal states. For the sake of convenience in a number of proofs, we assume that $Q \subsetneq \{q_i : i \in \mathbb{N}\}$.
- $q_{\mathrm{s}} \in Q$ is a special state called the initial state.
- $q_{\mathrm{lim}} \in Q$, with $q_{\mathrm{lim}} \neq q_{\mathrm{s}}$, is a special state called the limit state.
- $q_{\mathrm{f}} \in Q$, with $q_{\mathrm{f}} \neq q_{\mathrm{s}}$ and $q_{\mathrm{f}} \neq q_{\mathrm{lim}}$, is a special state called the final, or halting, state.

- $\delta : (Q \backslash \{q_{\mathrm{f}}\}) \times \{0,1\}^3 \to Q \times \{0,1\}^3 \times \{L, R\}$ is a (total) function, called the transition function. L and R here are fixed objects standing for *left* and *right* respectively.

Unlike regular Turing machines, which represent functions on the natural numbers, ITTMs will represent functions over subsets of $\mathbb{N}$, which in this thesis will be called 'reals'. We will now define the operations on an infinite time Turing machine using ordinal recursion. Four different definite operations will be defined:

- $q_\alpha^T(x)$ denotes the state a machine T is in at stage β, having started from the input x.
- $h_\alpha^T(x)$ denotes the position of the head at stage α, having started from the input x.
- $c_{i,\alpha}^T(x)$, where $i \in \{1,2,3\}$, denotes the content of tape i at stage α, having started from the input x. Also, we will define another operation, $c_{i,n,\alpha}^T(x)$, which will be defined as 1 if $n \in c_{i,\alpha}^T(x)$, and 0 otherwise.
- Of the three tapes we have, the first one will be called, and play the role of **input tape**, and the second one will be used as **output tape**.
- We will regularly use the term **the snapshot at** α to refer to the tuple of values $(q_\alpha^T(x), h_\alpha^T(x), c_{1,\alpha}^T(x), c_{2,\alpha}^T(x), c_{3,\alpha}^T(x))$.

Definition 2.2. For any infinite time Turing machine T and any input $x \in \mathbb{R}$, we define the operations $q_\alpha^T(x)$, $h_\alpha^T(x)$, $c_{i,\alpha}^T(x)$ as follows for all ordinals α, using transfinite recursion:

- If $\alpha = 0$, we set $q_\alpha^T(x) = q_{\mathrm{s}}$, $h_\alpha^T(x) = 0$, $c_{1,\alpha}^T(x) = x$, and $c_{i,\alpha}^T(x) = \emptyset$ for $i \in \{2,3\}$.
- If there is some ordinal $\beta < \alpha$ such that $q_\beta^T(x) = q_{\mathrm{f}}$, then $q_\alpha^T(x)$, $h_\alpha^T(x)$, and $c_{i,\alpha}^T(x)$ for $i \in \{1,2,3\}$ are undefined.
- Otherwise, if α is a successor ordinal $\beta + 1$, and $h_\beta^T(x) = h$, we look at the value of the transition function

  $$(q', (a_1, a_2, a_3), \Delta) = \delta(q_\beta^T(x), (c_{1,h,\beta}^T(x), c_{2,h,\beta}^T(x), c_{3,h,\beta}^T(x)))$$

  and set:

  - $q_\alpha^T(x) = q'$
  - $h_\alpha^T(x) = h + 1$ if $\Delta = R$; $h_\alpha^T(x) = h - 1$ if $\Delta = L$ and $h > 0$; and $h_\alpha^T(x) = h$ if $\Delta = L$ and $h = 0$.
  - For each $i \in \{1,2,3\}$, $c_\alpha^T(x) = c_\beta^T(x) \backslash \{h\}$ if $a_i = 0$, and $c_\beta^T(x) \cup \{h\}$ if $a_i = 1$.
- Otherwise, if α is a limit ordinal, we set $q_\alpha^T(x) = q_{\lim}$, $h_\alpha^T(x) = 0$, and for each $i \in \{1,2,3\}$, $c_\beta^T(x) = \{n \in \mathbb{N} : \limsup_{\alpha<\beta} c_{i,n,\alpha}^T(x) = 1\}$.

Taking into account the given interpretations of the operations q^T, h^T, and c^T, Definition 2.2 can be read as follows: to perform an infinite time Turing machine computation with a machine T on the real x, we start at the initial state q_s, with the head at position 0, with the real x on the first tape, and the second and third tapes still empty. At successor ordinals, just like in the case of regular Turing machines, we read the content each of the tapes at the current head position, and find the appropriate transition using the function δ from the current state q we are in, and the content a_1, a_2, a_3 of the tape cells we are in. The value of $\delta(q, (a_1, a_2, a_3))$ now determines to which cell we move, what we should write on the tape cells we just read, and whether to move the head left or right. At limit ordinal stages, the head is set to position 0, the machine will be in the special limit state $q_{\lim}$, and the content of each of the cells will be the lim sup of the content of that cell at all earlier stages.

Like usual, we can, because infinite time Turing machines themselves are hereditarily finite, assume the existence of a bijective coding function π that assigns to each infinite time Turing machine T an unique natural number e.

We can now introduce the notion of halting for infinite time Turing machines, and the notion of a function computed by an infinite time Turing machine:

Definition 2.3. We say that an infinite time Turing machine T **halts**, or **terminates**, on input x if there is some ordinal α such that $q_\alpha^T(x) = q_f$.

Definition 2.4. For any infinite time Turing machine T, we define the (partial) function ϕ_T as follows:

$$\phi_T = \{(x, y) \in \mathbb{R}^2 : \exists\alpha(\mathrm{Ord}(\alpha) \wedge q_\alpha^T(x) = q_f \wedge c_{2,\alpha}^T(x) = y)\}$$

Because there can, as a result of the definition, only be one ordinal α such that $q_\alpha^T(x) = q_f$, there can for any x be at most one y such that $(x, y) \in \phi_T$.

Furthermore we will, as usual, write $\phi_T(x) \downarrow$ if $x \in \mathrm{dom}(\phi_T)$, and $\phi_T(x) \uparrow$ if $x \notin \mathrm{dom}(\phi_T)$. It follows directly from the definitions that $\phi_T(x) \downarrow$ is true if and only if T halts on input x.

Definition 2.5. We say that a function f is computable by an infinite time Turing machine if there is some infinite time Turing machine T, such that $f = \phi_T$.

We now introduce a simple variation on the ITTM_0 model:

Definition 2.6. An ITTM_0 *with stay option* is defined identically to the ITTM_0 model, with the following exceptions:

1. δ now is a function from $(Q \backslash q_f) \times \{0, 1\}^3$ to $Q \times \{0, 1\}^3 \times \{L, R, 0\}$
2. In the definition of the operation $h_\alpha^T(w)$ at successor ordinals $\beta + 1$, if we again have that $h_\beta^T(x) = h$ and

$$(q', (a_1, a_2, a_3), \Delta) = \delta(q_\beta^T(x), (c_{1,h,\beta}^T(x), c_{2,h,\beta}^T(x), c_{3,h,\beta}^T(x)))$$

   we add the condition that, if $\Delta = 0$, then $h_\alpha^T(x) = h$.

Proposition 2.7. *For any function f, f is computable by an* ITTM_0 *with stay option if and only if it is computable by an* ITTM_0.

Proof. Say, the function f is computed by the infinite time Turing machine T with stay option, so $f = \phi_T$. We now construct a new infinite time Turing machine, T', in the following way:

1. Say, $k = \max\{i : q_i \in Q\}$. We now set $Q' = Q \cup \{q_{i+k} : q_i \in Q\}$. In other words, the new set of states Q' will contain all of the states in Q as well as, for every state $q_i \in Q$, another state q_{i+k}.

2. The special states $q_\mathrm{s}, q_\mathrm{lim}$, and q_f of T' refer to the same elements in Q' to which they referred in Q in T.

3. We construct δ' from δ as follows: for any $q_i \in Q$, and any $a \in \{0,1\}^3$, such that the third component of $\delta(q_i, a)$ is L or R, $\delta'(q_i, a)$ will be equal to $\delta(q_i, a)$. For any $q_i \in Q$, and any $a \in \{0,1\}^3$, such that $\delta(q_i, a)$ is equal to $(q_j, b, 0)$, we set $\delta'(q_i, a)$ to (q_{k+j}, b, R). Finally, for any q_{k+i}, i.e. any element of $Q'\backslash Q$, we set $\delta'(q_{k+i}, a) = (q_i, a, L)$.

In other words, for every existing state q_i, we have constructed a new state q_{k+i} such that, if we ever find ourselves in q_{k+i} during the computation, the machine will leave the tape content as it is, move the head left, and go on to state q_i. The new machine T' thus simulates the stay option in a move from q_i to q_j by first moving the head right, changing the tape content accordingly, and going on to state q_{j+k} and then doing nothing except for going left again, and going to q_j. It should be clear that this new machine indeed simulates the stay option.

For the other direction, the result is immediate, as any standard infinite time Turing machine also fulfils the definition of an infinite time Turing machine with stay option. □

Analogues of this proposition will also work for all of the other variations of infinite time Turing machines that will be introduced later on.

2.2 ITTM_0-machines with variations in the tapes

In the model defined in section 2.1, we defined infinite time Turing machines as consisting of exactly three tapes. One might wonder if the computational abilities of infinite time Turing machines are somehow affected if we use less tapes, or extra tapes; and/or if we add extra scratch cells that are not part of any tape. Many of these questions have been dealt with in [HaSe], with a number of equivalence results, and one striking non-equivalence. To begin with, we need to make precise what we mean with 'an infinite time Turing machine with n tapes and m scratch cells':

Definition 2.8. For any $m \geq 1$, and any $n \geq 0$, ITTM_0s with m tapes and n scratch cells (or, as we will alternately call them, 'flags') will be defined identically to the model defined in section 2.1, with the following exceptions: the transition function δ will now be a function from $Q \times \{0,1\}^{m+n}$ to $Q \times \{0,1\}^{m+n} \times \{L, R\}$. We again recursively define the operation $c^T_{i,\alpha}(x)$ for the content of tape i at stage α on a computation starting from x, and now introduce a new operation $f^T_{i,\alpha}(x)$ for the content of the ith scratch cell at stage α on the computation starting from x. The operation at the successor ordinals will now

take into account the contents of all the tapes at the position of the head, as well as the scratch cells (which are unaffected by the position of the head), and again move left or right. The content of all the cells on the tapes at limit stages, and all scratch cells, will again be defined as the lim sup of the earlier stages of those cells.

In the special case where $m = 1$, the first (and only) tape will take over the role of output tape; in all other cases, the second tape will be the output tape, as in the earlier definition. Whenever we simply mention 'an ITTM_0 with m tapes', we mean an ITTM_0 with m tapes without any scratch cells. We also define ITTM_0s with m tapes and n scratch cells and a stay option like in the three-tape case.

Proposition 2.7 generalizes to machines with m tapes and n scratch cells:

Proposition 2.9. *For any m and any n, a function f is computable by an ITTM_0 with m tapes and n scratch cells if and only if it is computable by an ITTM_0 with m tapes and n scratch cells and a stay option.*

Proof. The proof of Proposition 2.7 works if we replace each occurrence of $\{0,1\}^3$ with $\{0,1\}^{m+n}$. □

Also, it is easy to see that scratch cells can be simulated with two extra tapes:

Proposition 2.10. *If a function f is computable by an ITTM_0 T with n tapes and $m \geq 1$ scratch cells, then it is also computable by an ITTM_0 T' with $n+2$ tapes and $m-1$ scratch cells.*

Proof. We use the first cell of the $n+2$nd tape to simulate the scratch cell. Furthermore, at the first ω stages of the computation, we fill the $n+1$st tape with 1s. At stage ω, we read a 1 at the first cell of the $n+1$st tape while in the limit state, and replace it again by a 0, never to change it back afterwards, and start the actual computation. Whenever we need to access the scratch cell, we write a 0 on the $n+1$st tape at the current head position, move left until we see the other 0 on that tape, access the scratch cell and perform any necessary computations, move right again until we are back at the 0 signifying the head position, and replace it again with a 1. When we find ourselves in the limit state reading a 0 on the second tape, we just go on with the simulation, with the knowledge that this situation corresponds to the simulated machine being at a limit state.

In the case where $n = 1$, we furthermore need to make sure that the output of the three-tape simulation of the one-tape machine appears on the second tape. To do this, in the first ω steps we copy the contents of the first tape to the second tape while filling the first tape with 1s. From point ω onward, we then use the first tape to hold information about the current head position and the beginning of the tape, and use the second tape to simulate the contents of our original tape. □

With this definition, we can now turn to the following theorem, one of the main theorems of [HaSe]:

Proposition 2.11. *For any f, f is computable by a regular ITTM_0 (with three tapes and no scratch cells) if and only if f is computable by an ITTM_0 with one tape and one scratch cell.*

Proof. [HaSe, Theorem 2.3] shows us that an ITTM_0 with only one tape and one scratch cell can compute any function that is computable by a standard ITTM_0. For the other direction, Proposition 2.10 shows us that any function that is computable by an ITTM_0 with one tape and one scratch cell is also computable by an ITTM_0 with three tapes. □

Proposition 2.12. *For any $n \geq 2$ and any function f, f is computable by an ITTM_0 with n tapes if and only if it is computable by a regular ITTM_0.*

Proof. See [HaSe, Corollary 2.4]. □

We can obtain an even stronger, and more general, result:

Proposition 2.13. *For any n and any m, where $n + m \geq 2$ and $m \geq 1$, and any function f, f is computable by an ITTM_0 with m tapes and n scratch cells if and only if it is computable by a regular ITTM_0.*

Proof. First, note that it is either true that $n \geq 1$ and $m \geq 1$, in which case it follows from Proposition 2.11 that every function computable by a standard ITTM_0 can be computed by an ITTM_0 with n scratch cells and m tapes, simply by ignoring any extra tapes we might have; or it is true that $m \geq 2$, in which case the same argument can be applied to the result of Proposition 2.12.

We still need to show that for any such n and m, a function that is computable by an ITTM_0 with n scratch cells and m tapes can also be computed by a normal ITTM_0. Here, repeatedly applying Proposition 2.10 gives us that an ITTM_0 with $m + 2n$ tapes and no scratch cells can compute every function f that is computable by an ITTM_0 with n scratch cells and m tapes. Now Proposition 2.12 yields us the result that f can also be computed by a standard ITTM_0 with three tapes. □

In stark contrast to the above equivalence results, we have the following surprising difference between one-tape ITTM_0s, and three-tape ITTM_0s:

Proposition 2.14. *The set of functions computable by ITTM_0s with only tape is strictly smaller than the set of functions computable by ITTM_0s with 3 tapes.*

Proof. See [HaSe, Theorem 2.1]. □

In [HaSe], it is further noted that the set of functions computable by ITTM_0s with only one tape is actually only slightly smaller than the set of functions computable by a regular ITTM_0: it is shown that, for any function f, if f is computable by an ITTM_0, then $1 * f$ (that is, the function g, such that $1 \in g(x)$ for all x, and $x + 1 \in g(x)$ if and only if $x \in f(x)$) is computable by an ITTM_0 with only one tape. The main issue at hand here is that the one-tape machines are not closed under composition. To see that this must be the case, consider the following argument: if they were closed under composition, we could of course first compute $1 * f$, and compose this function with the 'move everything left one state' function (which is easily computable even by a one-tape machine) in order to obtain f. But there are cases where $1 * f$ is computable by a one-tape machine while f is not, so we cannot have closure under composition. The issue at hand here really seems to be a matter of a lack of space.

In summary, we find out that we can change the number of tapes, and the number of scratch cells, in a large number of ways. Most importantly, we can,

when giving a computation, always add scratch cells and extra tapes at will, without fundamentally affecting the computational power of infinite time Turing machines. Just about the only thing that cannot be done without causing any problems, is reducing the number of tapes to one without adding any scratch cells.

2.3 ITTM_1: different rules at limit points

Other than changing the number of tapes and possibly adding one or more scratch cells, we can also conceive of other modifications in the infinite time Turing machine architecture: for example, we may consider different operations at limit states. In this section, we will introduce the ITTM_1-architecture, in which we get rid of the special limit state, and instead let the state $q_\alpha^T(w)$ at limit ordinals α be determined by the history of earlier visited states.

Definition 2.15. An ITTM_1 machine and its operations are defined identically to the earlier ITTM_0 machines, with the following differences:

- There is no limit state q_{lim}.
- During the computation from w by a machine T, if we are at a limit stage α, the current state $q_\alpha^T(w)$ will be defined as the lim sup of all the earlier states $q_\beta^T(w) : \beta < \alpha$. Because of this extra requirement, that depends on the numerical value of the states, it becomes essential that the earlier requirement that elements of Q can always be identified by natural numbers, indeed holds.

Again, we define ITTM_1 machines with stay options, and ITTM_1 machines with n tapes and m scratch cells analogously to how these variants were defined in the ITTM_0 case.

Also for these ITTM_1 machines, and the variants with n tapes and m scratch cells, an analogue of Proposition 2.7 holds true:

Proposition 2.16. *For any m and any n, a function f is computable by an ITTM_1 with m tapes and n scratch cells if and only if it is computable by an ITTM_1 with m tapes and n scratch cells and a stay option.*

Proof. The construction from Proposition 2.7 here also works, if we replace each occurrence of $\{0,1\}^3$ with $\{0,1\}^{m+n}$ and leave out the clause about the limit state. We just have to make sure that we rename the original states so that they remain in the same order as they were, and that they all have higher identifying numbers than the newly made states. Because of this, at a limit stage, we always will find ourselves in the 'correct' state corresponding to the set of original states, and the simulation will be carried out as intended. □

Beyond this, on a first glance, this modification seems to be quite restrictive, and not to make things easier in any way: whereas in the earlier defined ITTM_0 machines a program could always 'see' whether it was in a limit state or not, we now seem to have lost this ability, because we have gotten rid of our limit states. As of yet, we have not yet proven that this is indeed the case: we may try to tackle this problem by using special 'signal' states and/or special flags

to detect limit states. Of course, just like in the case of ITTM_0, we first need to establish that we can add flags and extra tapes, while preserving the same range of computable functions.

2.4 Equivalence results for ITTM_1 and variants thereof

As it turns out, in the case of ITTM_1 machines, we can again prove a number of equivalence results, taking a significant amount of inspiration from the theorems in [HaSe]. We start out by defining the computable join $\oplus$ of two reals x and y as usual:

Definition 2.17. If x and y are both reals, we say that:

$$x \oplus y = \{2n : n \in x\} \cup \{2n+1 : n \in y\}$$

We also need a notion that is more general than the earlier defined notion of a functional computation on an infinite time Turing machine, which allows us to consider operations by infinite time Turing machines from n-tape inputs to n-tape outputs, rather than only functions on the reals:

Definition 2.18. We define an operation on an n-tape infinite time Turing machine from $x_1, \ldots, x_n$, using the definite operations c^T, q^T, h^T, by setting $c^T_{i,0}(x_1, \ldots, x_n) = x_i$ for all $i \in \{1, \ldots, n\}$, and then proceeding as in the case of a computation from a single real. The operations will be described by functions ψ_T, such that $\psi_T(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ if α is the ordinal such that $q^T_{i,\alpha}(x_1, \ldots, x_n) = q_{\mathrm{f}}$, and $c^T_{i,\alpha}(x_1, \ldots, x_n) = y_i$ for all $i \in \{1, \ldots, n\}$, and $\psi_T(x_1, \ldots, x_n) \uparrow$ if there is no ordinal satisfying this requirement.

With this definition, we will first provide several lemmata, all establishing the existence of certain operations for ITTM_1s with n tapes. After that, we return to the question whether ITTM_1s with varying numbers of tapes have equivalent computing powers.

On a first glance, even a very simple operation like moving the tape contents right one cell, seems problematic. A machine that spends the first ω steps of the computation moving the tape contents right one cell is easily constructed, but we also need to be able to recognize when this operation is done. For this, in the next proof, and in many of the following ones in this chapter, we will make use of a technique that enables us to recognize limit states.

Lemma 2.19. *For any $n \geq 1$, there is an ITTM_1 T with n tapes, computing an operation ψ such that, for all reals $x_1, \ldots, x_n$, $\psi(x_1, \ldots, x_n) = (\{k+1 : k \in x_1\}, \ldots, \{k+1 : k \in x_n\})$.*

Proof. The general idea is to make use of a number of special 'signal states' in the set Q, and construct the machine so that we are certain that, if one of these signal states is visited at a successor ordinal stage, it will always read one thing on the tape, whereas, if one of these states is visited at a limit ordinal stage, it will read another thing. In this proof, for example, we know that at position 0 of the tapes, we have the content $(0, \ldots, 0)$, and construct the machine so that, if a signal state is visited at a successor ordinal stage, it will always read a 1

on the first tape. Moreover, we can make sure that, at a limit ordinal stage, we will always find ourselves in one of the signal states by making sure that at least one of these stages will be visited cofinally often, and by making sure that all signal states have higher identifying numbers than non-signal states.

Having established this general technique, we can now proceed with the actual computation of the operation in the following way. The set Q of our machine T will contain the following states:

- For each $(a_1, \ldots, a_n) \in \{0,1\}^n$, a state $q[A, a_1, \ldots, a_n]$.
- For each $b \in \{0,1\}$, and each $(a_1, \ldots, a_n) \in \{0,1\}^n$, a signal state $q[\mathrm{sig}_A, b, a_1, \ldots, a_n]$.
- An initial state q_s, and a final state q_f, separate from the other states.

We furthermore ensure that the signal states are all assigned the highest identifying numbers: because of this, we can be certain that, if at a limit stage both signal states and non-signal states have been visited cofinally often, we will find ourselves in one of the signal states.

We now define the transition function δ as follows. For all $(a_1, \ldots, a_n) \in \{0,1\}^n$, and all $(b_1, \ldots, b_n) \in \{0,1\}^n$, we define:

$$\begin{gathered}
\delta(q_\mathrm{s}, (a_1, \ldots, a_n)) = (q[A, a_1, \ldots, a_n], (0, \ldots, 0), R) \\
\delta(q[A, a_1, \ldots, a_n], (b_1, \ldots, b_n)) = (q[\mathrm{sig}_A, a_1, b_1, \ldots, b_n], (1, a_2, \ldots, a_n), 0) \\
\delta(q[\mathrm{sig}_A, a_1, b_1, \ldots, b_n], (1, a_2, \ldots, a_n)) = (q[A, b_1, \ldots, b_n], (a_1, \ldots, a_n), R) \\
\delta(q[\mathrm{sig}_A, a_1, \ldots, a_n], (0, \ldots, 0)) = (q_\mathrm{f}, (0, \ldots, 0), 0)
\end{gathered}$$

It should be noted that this function is not in fact total on the domain $(Q \backslash q_\mathrm{f}) \times \{0,1\}^n$: δ is not defined on $q[\mathrm{sig}_A, a_1, \ldots, a_n], (0, b_2, \ldots, b_n)$ if one or more of the b_is are 1 whereas. We can however extend this δ in any way we like to a total function, and the precise way of doing so will be irrelevant, as these transitions will never occur in the computation. This remark can be extended to most of the later proofs where complete descriptions of infinite time Turing machines are given.

It is fairly easy to see that this machine performs the desired operation. Starting from state q_s, we will first visit a state from the A-group, then a state from the sig_A-group, then again a state from the A-group, and so on, each time moving the content of the tape at the current position of the head right one step. Also, each time we are in a state from the sig_A-group at a stage below ω, we will read a 1 on the first tape, as a direct result of the construction.

At stage ω, we will once again find ourselves in a state from the sig_A-group, but this time we read a 0 on the first tape, signifying that we are done with the operation, and we move to the final state. □

Lemma 2.20. *For any $n \geq 1$, there is an* ITTM_1 *T with n tapes, computing an operation ψ such that, for all reals $x_1, \ldots, x_n$, $\psi(x_1, \ldots, x_n) = (\{2k : k \in x_1\}, \ldots, \{2k : k \in x_n\})$ as long as $0 \in x_1$.*

Proof. Consider a machine T where the set Q contains the following states:

- For each $(a_1, \ldots, a_n) \in \{0,1\}^n$, a state $q[B, a_1, \ldots, a_n]$.

- For each $b \in \{0,1\}$, and for each $(a_1,\ldots,a_n) \in \{0,1\}^n$, a signal state $q[\mathrm{sig}_B, b, a_1, \ldots, a_n]$.
- States $q[A]$, $q[C]$, $q[D]$, and $q[E]$, another signal state $q[\mathrm{sig}_C]$, and two states q_s and q_f separate from all the other states.

Of all these states, we make sure that the signal states are assigned higher identifying numbers than the rest of the states, and that $q[\mathrm{sig}_C]$ is, in turn, assigned a higher identifying number than all other signal states. Beyond this, the exact ordering of the states does not matter.

We can now define the transition function δ as follows. For all $(a_1,\ldots,a_n) \in \{0,1\}^n$, and all $(b_1,\ldots,b_n) \in \{0,1\}^n$, we set:

$$\delta(q_\mathrm{s}, (a_1,\ldots,a_n)) = (q[A], (a_1,\ldots a_n), R)$$
$$\delta(q[A], (a_1,\ldots,b_n)) = (q[B, a_1,\ldots,a_n], (1,\ldots,1), R)$$

$$\delta(q[B, a_1,\ldots,a_n], (b_1,\ldots,b_n)) = (q[\mathrm{sig}_B, a_1, b_1,\ldots,b_n], (0, a_2,\ldots,a_n), 0)$$
$$\delta(q[\mathrm{sig}_B, a_1, b_1,\ldots,b_n], (0, a_2,\ldots,a_n)) = (q[B, b_1,\ldots,b_n], (a_1,\ldots,a_n), R)$$
$$\delta(q[\mathrm{sig}_B, a_1, b_1,\ldots,b_n], (1, a_2,\ldots,a_n)) = (q[C], (1, a_2,\ldots,a_n), R)$$
$$\delta(q[C], (0,\ldots,0)) = (q[\mathrm{sig}_C], (0,\ldots,0), 0)$$
$$\delta(q[C], (1,\ldots,1)) = (q[E], (0,\ldots,0), R)$$
$$\delta(q[\mathrm{sig}_C], (0,\ldots,0)) = (q[D], (0,\ldots,0), R)$$
$$\delta(q[\mathrm{sig}_C], (1, a_2,\ldots,a_n)) = (q_\mathrm{f}, (1, a_2,\ldots,a_n), R)$$
$$\delta(q[D], (a_1,\ldots,a_n)) = (q[C], (a_1,\ldots,a_n), R)$$
$$\delta(q[E], (a_1,\ldots,a_n)) = (q[A], (a_1,\ldots,a_n), R)$$

It now remains to be seen that this machine indeed performs the desired computation. For this, we will make a number of claims, which in combination establish the desired result.

Sublemma. *With T, n, and $x_1,\ldots,x_n$ as above, and $0 \in x_1$, the following are true:*

1. *For each $k \in \mathbb{N}$ with $k > 0$, and each $i \in \{1,\ldots,n\}$, $c^T_{i,\omega\cdot k}(x_1,\ldots,x_n) = \{2m : m < k \wedge m \in x_i\} \cup \{2k-1\} \cup \{m+k : m \geq k \wedge m \in x_i\}$; and $q^T_{\omega\cdot k}(x_1,\ldots,x_n)$ is a state from the sig_B-group.*
2. *For each $k \in \mathbb{N}$ with $k > 0$, the cells at positions with index $i \in \{0,\ldots,2k-2\}$ do not change between stage $\omega\cdot k$ and $\omega\cdot(k+1)$.*
3. *$q^T_{\omega^2}(x_1,\ldots,x_n) = q[\mathrm{sig}_C]$, and for each $i \in \{1,\ldots,n\}$, $c^T_{i,\omega^2}(x_1,\ldots,x_n) = \{2m : m \in x_i\}$.*

Proof. To start, we observe that there is no transition defined in δ that ever moves the head left, and that the two transitions that do not move the head will always be followed by another transition that moves the head right. Because of this, it follows, for any $k, m \in \mathbb{N}$, that $h^T_{\omega\cdot k+2m} \geq m$, and moreover, that $c^T_{i,p,\omega\cdot k+2m} = c^T_{i,p,\omega\cdot(k+1)}$ for all $p < m$ and all $i \in \{1,\ldots,n\}$.

We will prove the first claim by induction, first showing the base case where $k = 1$, and then proving the case $k = m+1$ from the assumption that the claim holds for all $k \in \{1,\ldots,m\}$.

From the initial state, the machine will directly move to state $q[A]$, where it will write down 1s on all tapes at position 1. From there, the computation continues with the sequence $q[B,\ldots] \to q[\mathrm{sig}_B,\ldots] \to q[B,\ldots]\ldots$, moving the content of each position of each tape, starting with index 1, right one step. Because the set of all states in the sig_B-group is visited cofinally often before ω, and because it is a finite set of states, at least one of these states is visited cofinally often. The only other states that might be visited cofinally often before ω are the states in the B-group: however, because they have a lower identifying number than the states in the sig_B-group, it is certain that q^T_ω is a state from the sig_B-group. As for the content of the tapes at stage ω, it follows that $c^T_{i,\omega} = \{2m : m < 1 \wedge m \in x_i\} \cup \{1\} \cup \{m+1 : m \geq 1 \wedge m \in x_i\}$, from which it is immediate that the first claim holds for the case $k = 1$.

Now assume that the first claim holds for all elements of $\{1,\ldots,k\}$. We thus know that $q^T_{\omega\cdot k}$ is a state from the sig_B-group. Now let p be equal to $k+1$. From $c^T_{i,\omega\cdot k} = \{2m : m < k \wedge m \in x_i\} \cup \{2k-1\} \cup \{m+k : m \geq k \wedge m \in x_i\}$ it follows, that for any r with $r < k$, all the cells at position $2r-1$ are 0 at stage $\omega \cdot k$. Now the computation will go from some cell from the sig_B-group at stage $\omega \cdot k$, where it reads a 1 (because $0 \in x_1$), to $q[C]$, and alternate between $q[C]$, $q[\mathrm{sig}_C]$, and $q[D]$ until it reads a sequence of ones at position $2k-1$. At this point, the 1s will be erased, and the head will be moved two more steps to the right, leaving the content at position $2k$ intact. Here, at position $2k+1$, a new sequence of 1s will be written down, and like in the first ω steps, the computation continues by moving the content of every cell from position $2k+1$ onward one cell to the right, while visiting only states from the B-group and states from the sig_B-group. It thus follows again, that $q^T_{\omega\cdot(k+1)}$ is a state from the sig_B-group. Moreover, we have that $c^T_{i,\omega\cdot(k+1)} = \{2m : m < k \wedge m \in x_i\} \cup \{2k \text{ if } k \in x_i\} \cup \{2k+1\} \cup \{m+k+1 : m \geq k+1 \wedge m \in x_i\}$, which is easily identical to $\{2m : m < k+1 \wedge m \in x_i\} \cup \{2(k+1)-1\} \cup \{m+k+1 : m \geq k+1 \wedge m \in x_i\}$. So the first claim also holds for the case $p = k+1$, and hence must hold for all $k \in \mathbb{N}$ with $k > 1$. For the second claim, it is easy to see that, during the computation between $\omega \cdot k$ and $\omega \cdot k + 1$, the head is first moved towards position $2k-1$ without changing anything. Because the head never moves left, it follows that the cells at positions indexed $\{0,\ldots,2k-2\}$ are not changed during this part of the computation.

For the third claim, note that $q^T_{\omega\cdot k+2} = q[\mathrm{sig}_C]$ for all $k > 2$, so $q[\mathrm{sig}_C]$ is visited cofinally often before ω^2, and hence $q^T_{\omega^2} = q[\mathrm{sig}_C]$. Furthermore, the second claim implies that, for each $k \in \mathbb{N}$ with $k > 0$, the cells at positions with index $i \in \{0,\ldots 2k-2\}$ do not change between stage $\omega \cdot k$ and ω^2. Thus, the value of the cells at ω^2 will be equal to the stabilized positions, and it is easy to see that this implies that $c^T_{i,\omega^2}(x_1,\ldots,x_n) = \{2k : k \in x_i\}$. □

It follows directly from the above sublemma, that T performs the desired operation ψ. □

Lemma 2.21. *For any $n \geq 1$, there is an* ITTM_1 *T with n tapes, computing an operation ψ such that, for all reals $x_1,\ldots,x_n$, $\psi(x_1,\ldots,x_n) = (\{k : k+1 \in x_1\},\ldots,\{k : k+1 \in x_n\})$.*

Proof. Consider a machine T where the set Q contains the following states:

- For each $(a_1, \ldots, a_n) \in \{0,1\}^n$, a state $q[0, B, a_1, \ldots, a_n]$ and a state $q[1, B, a_1, \ldots, a_n]$.
- A state $q[A]$, a state $q[0, C]$, a state $q[1, C]$, and two signal states $q[0, \mathrm{sig}_B]$ and $q[1, \mathrm{sig}_B]$.
- An initial state q_s and a final state q_f, both separate from the other states.

Again, we will ensure that the signal states $q[0, \mathrm{sig}_B]$ and $q[1, \mathrm{sig}_B]$ are assigned the highest identifying numbers. We now define the transition function δ as follows. For all $(a_1, \ldots, a_n) \in \{0,1\}^n$, and all $(b_1, \ldots, b_n) \in \{0,1\}^n$, we set:

$$\begin{aligned}
\delta(q_\mathrm{s}, (a_1, \ldots, a_n)) &= (q[A], (a_1, \ldots, a_n), R) \\
\delta(q[A], (0, a_2, \ldots, a_n)) &= q([0, B, 0, a_2, \ldots, a_n], (1, \ldots, 1), L) \\
\delta(q[A], (1, a_2, \ldots, a_n)) &= q([1, B, 1, a_2, \ldots, a_n], (0, \ldots, 0), L) \\
\delta(q[0, B, a_1, \ldots, a_n], (b_1, \ldots, b_n)) &= (q[0, \mathrm{sig}_B, (a_1, \ldots, a_n), R)
\end{aligned}$$

$$\begin{aligned}
\delta(q[1, B, a_1, \ldots, a_n], (b_1, \ldots, b_n)) &= (q[1, \mathrm{sig}_B, (a_1, \ldots, a_n), R) \\
\delta(q[0, \mathrm{sig}_B], (1, \ldots, 1)) &= (q[0, C], (1, \ldots, 1), R) \\
\delta(q[0, \mathrm{sig}_B], (0, a_2, \ldots, a_n)) &= (q_\mathrm{f}, (0, a_2, \ldots, a_n), 0) \\
\delta(q[1, \mathrm{sig}_B], (0, \ldots, 0)) &= (q[1, C], (0, \ldots, 0), R) \\
\delta(q[1, \mathrm{sig}_B], (1, a_2, \ldots, a_n)) &= (q_\mathrm{f}, (1, a_2, \ldots, a_n), 0) \\
\delta(q[0, C], (a_1, \ldots, a_n)) &= (q[0, B, a_1, \ldots, a_n], (1, \ldots, 1), L) \\
\delta(q[1, C], (a_1, \ldots, a_n)) &= (q[1, B, a_1, \ldots, a_n], (0, \ldots, 0), L)
\end{aligned}$$

We now still need to check that this machine performs the desired computation. The computation always starts in the state q_s, and there moves the head right one cell without affecting the tape content, while going to state $q[A]$. With the head at position 1, in $q[A]$, we distinguish two possibilities: either we read a 1 on the first tape, or we read a 0.

In the case where we read a 1 on the first tape, we go to a state from the $q[1, B, \ldots]$-group that codes the content that has just been read, move the head left again, while writing a sequence of 0s on the position of the tape with index 1. So, at stage 2, we find ourselves in a state from the $q[1, B, \ldots]$ group that codes the original content of position 1 of the tapes, with the head at position 0. From this point onward, we can show by induction on the natural numbers, that for each k: (1) at stage $3k + 2$ we are in some state from the $q[1, B, \ldots]$-group that codes the original content of the tapes at position $k + 1$, with the head at position k, where we write the content that originally was at position $k + 1$ of the tape, and move right; (2) at stage $3k + 3$ we are in state $q[1, \mathrm{sig}_B]$ with the cell at position $k + 1$, reading 0s that were written there at stage $3k + 1$, and move right again; and (3) at stage $3k + 4$ we are in state $q[1, C]$ with the cell at position $k + 2$, reading the original content of the tape at this position, replacing it with a sequence of 0s, and moving left again. Moreover, because we alternate between moving right twice, at stages $3k + 2$ and $3k + 3$, and moving left once, at stage $3k + 4$, it follows that the cells at any position $h \in \mathbb{N}$ never change between stage $3k + 2$ and stage ω. As a result of all this, at stage ω for every $k \in \mathbb{N}$, the original contents of the cells at position $k + 1$ have been

written at position k. At stage ω, finally, we will be in state $q[1, \mathrm{sig}_B]$, where we read a 1 on the first tape, and move to the final state.

In the case we read a 0 on the first tape, things go very similar, with the difference that $q[0, \mathrm{sig}_B]$ now reads 1s in successor states, that $q[0, C]$ writes 1s instead of 0s, and that $q[0, \mathrm{sig}_B]$ will read a 0 on the first tape at stage ω. □

Lemma 2.22. *For any $n \geq 1$ and any* ITTM_1 *T with $2n$ tapes, computing the operation ψ_T, there is another* ITTM_1 *T', with n tapes, computing the operation $\psi_{T'}$, such that, for all reals $x_1, \ldots, x_{2n}$, if $\psi_T(x_1, \ldots, x_{2n}) = (y_1, \ldots, y_{2n})$, then $\psi_{T'}(x_1 \oplus x_{n+1}, \ldots, x_n \oplus x_{2n}) = (y_1 \oplus y_{n+1}, \ldots, y_n \oplus y_{2n})$.*

Proof. We can carry out this simulation by letting each of the n tapes of the new machine T' take the role of two of the tapes of the original machine T. In the simulation, the ith tape of the new machine T', for all $i \in \{1, \ldots, n\}$, will be used to store the contents of the ith as well as the $i+n$th tape of the original machine.

The simulation operation itself can be performed in the following way: say, T has a set of k internal states $Q = \{q_1, \ldots, q_k\}$, and $2n$ tapes. The new machine, T', with n tapes, will have the following set of states Q':

- For each $i \in \{1, \ldots, k\}$, Q' will contain a state $q[A, i]$.
- For each $(a_1, \ldots, a_n) \in \{0,1\}^n$ and each $i \in \{1, \ldots, k\}$, Q' will contain a state $q[B, i, a_i, \ldots, a_n]$.
- For each $(a_1, \ldots, a_n) \in \{0,1\}^n$, each $i \in \{1, \ldots, k\}$, and each $\Delta \in \{L, R\}$, Q' will contain a state $q[C, a_1, \ldots, a_n, i, \Delta]$.
- For each $i \in \{1, \ldots, k\}$ and each $\Delta \in \{L, R\}$, Q' will contain a state $q[D, i, \Delta]$.

Of all these states, we will say that the states $\{q[A,1], \ldots, q[A,k]\}$ 'correspond' to the original states, and make sure that they are assigned the highest identifying numbers of all the states, as well as that their identifying numbers preserve the ordering of the original set of states $\{q_1, \ldots, q_k\}$. Also, if the initial state of T is q_i, and the final state is q_j, we set the initial state of T' to $q[A, i]$, and the final state of T' to $q[A, j]$.

We now continue by defining the new transition function δ':

- For each $(a_1, \ldots, a_n) \in \{0,1\}^n$ and each $i \in \{1, \ldots, k\}$, we set:

$$\delta'(q[A, i], (a_1, \ldots, a_n)) = (q[B, i, a_1, \ldots, a_n], (a_1, \ldots, a_n), R)$$

- For each $(a_1, \ldots, a_n) \in \{0,1\}^n$, each $(b_1, \ldots, b_n) \in \{0,1\}^n$, and each $i \in \{1, \ldots, k\}$, we first look at

$$\delta(q_i, (a_1, \ldots, a_n, b_1, \ldots, b_n)) = (q_j, (c_1, \ldots, c_n, d_1, \ldots, d_n), \Delta)$$

and then set

$$\delta'(q[B, i, a_1, \ldots, a_n], (b_1, \ldots, b_n)) = (q[C, c_1, \ldots, c_n, j, \Delta], (d_1, \ldots, d_n), L)$$

- For each $(a_1, \ldots, a_n) \in \{0,1\}^n$, each $b \in \{0,1\}^n$, each $i \in \{1, \ldots, k\}$, and $\Delta \in \{L, R\}$, we set

$$\delta'(q[C, a_1, \ldots, a_n, i, \Delta], b) = (q[D, i, \Delta], (a_1, \ldots, a_n), \Delta)$$

- Finally, for each $a \in \{0,1\}^n$, each $i \in \{1, \ldots, k\}$, and each $\Delta \in \{L, R\}$, we set:

$$\delta'(q[D, i, \Delta], a) = (q[A, i], a, \Delta)$$

It still remains to be proven that this machine T' correctly performs the simulation of T. This will be done, using transfinite induction, in the following sublemma. We here use the shorthand x for the input reals $x_1, \ldots, x_{2n}$ of T, and the shorthand x' for the input reals $x_1 \oplus x_{n+1}, \ldots, x_n \oplus x_{2n}$ for the input reals of T'.

Sublemma. *The following claims hold for all ordinals α, with T, T', n, x, and x' as defined above:*

1. *If $q_\alpha^T(x) = q_k$, then $q_{4\cdot\alpha}^{T'}(x') = q[A, k]$.*
2. *$2 \cdot h_\alpha^T(x) = h_{4\cdot\alpha}^{T'}(x')$.*
3. *For all $i \in \{1, \ldots, n\}$, $c_{i,4\cdot\alpha}^{T'}(x') = c_{i,\alpha}^T(x) \oplus c_{n+i,\alpha}^T(x)$.*
4. *For all ordinals $\eta \leq 4 \cdot \alpha$, if $q[A, i]$ is visited at stage η, then η must be $4 \cdot \beta$ for some $\beta \leq \alpha$.*

Proof. (1) The case where $\alpha = 0$: this directly follows from the definitions and the chosen input of T and T' respectively.

(2) The case where α is $\beta + 1$: assume that the above claims hold for the case β, so $c_{i,4\cdot\beta}^{T'}(x') = c_{i,\beta}^T(x) \oplus c_{n+i,\beta}^T(x)$, $q_\beta^T(x) = q_k$ for some k, and $q_{4\cdot\beta}^{T'}(x) = q[A, k]$, and $2 \cdot h_\beta^T(x) = h_{4\cdot\beta}^{T'}(x')$. Using the shorthand notations $h = h_\beta^T(x)$ and $h' = h_{4\cdot\beta}^{T'}(x') = 2h$, and looking at the individual cells, we furthermore get $c_{i,\beta,h}^T(x) = c_{i,4\cdot\beta,h'}^{T'}(x')$, and $c_{n+i,\beta,h}^T(x) = c_{i,4\cdot\beta,h'+1}^{T'}(x')$ for all $i \in \{1, \ldots, n\}$.

Now, say, the transition function δ gives us:

$$\delta(q_k, (c_{1,\beta,h}^T(x), \ldots c_{2n,\beta,h}^T(x)) = (q_l, (a_1, \ldots, a_n, b_1, \ldots, b_n), \Delta))$$

From the definitions, it will now follow that:

$$\begin{aligned} & \delta'(q[A, k], (c_{1,4\cdot\beta,h'}^{T'}(x'), \ldots, c_{n,4\cdot\beta,h'}^{T'}(x'))) \\ = \; & \delta'(q[A, k], (c_{1,\beta,h}^T(x), \ldots, c_{n,\beta,h}^T(x))) \\ = \; & (q[B, k, c_{1,\beta,h}^T(x), \ldots, c_{n,\beta,h}^T(x)], (c_{1,\beta,h}^T(x), \ldots c_{n,\beta,h}^T(x)), R) \end{aligned}$$

Because the values of the cells do not change during this step, we will have $c_{i,4\cdot\beta+1,k}^{T'}(x') = c_{i,4\cdot\beta,k}^{T'}(x')$ for all i and k. Also, we will have, $h_{4\cdot\beta+1}^{T'}(x') = h' + 1$ for the head of the machine, and $q_{4\cdot\beta+1}^{T'}(x') = q[b, k, c_{1,\beta,h}^T(x), \ldots, c_{n,\beta,h}^T(x)]$ for the internal states. Looking at the next step of computation, we will get:

$$\begin{aligned} & \delta'(q[B, k, c_{1,\beta,h}^T(x), \ldots, c_{n,\beta,h}^T(x)], (c_{1,4\cdot\beta+1,h'+1}^{T'}(x'), \ldots, c_{n,4\cdot\beta+1,h'+1}^{T'}(x'))) \\ = \; & \delta'(q[B, k, c_{1,\beta,h}^T(x), \ldots, c_{n,\beta,h}^T(x)], (c_{n+1,\beta,h}^T(x), \ldots, c_{2n,\beta,h}^{T'}(x))) \\ = \; & (q[C, a_1, \ldots, a_n, l, \Delta], (b_1, \ldots, b_n), L) \end{aligned}$$

Now, the cells at position $h'+1$ have changed, so that $c^{T'}_{i,4\cdot\beta+2,h'+1}(x') = b_i$ for all $i \in \{1,\ldots,n\}$. For all $k \neq h'+1$ and all n, we again have $c^{T'}_{i,4\cdot\beta+2,k}(x') = c^{T'}_{i,4\cdot\beta,k}(x')$. Furthermore, we have $q^{T'}_{4\cdot\beta+2}(x') = q[C,a_1,\ldots,a_n,l,\Delta]$ as well as $h^{T'}_{4\cdot\beta+2}(x') = h'$. For the next step of computation:

$$\begin{aligned} & \delta'(q[C,a_1,\ldots,a_n,l,\Delta],(c^{T'}_{1,4\cdot\beta+2,h'}(x'),\ldots,c^{T'}_{n,4\cdot\beta+2,h'}(x'))) \\ = \;& \delta'(q[C,a_1,\ldots,a_n,l,\Delta],(c^{T}_{1,\beta,h}(x),\ldots c^{T}_{n,\beta,h}(x))) \\ = \;& (q[D,l,\Delta],(a_1,\ldots,a_n),\Delta) \end{aligned}$$

Now, the cells at position h' have also changed, so that $c^{T'}_{i,4\cdot\beta+3,h'}(x') = a_i$ for all $i \in \{1,\ldots,n\}$. For cells at position $h'+1$, we still have $c^{T'}_{i,4\cdot\beta+3,h'+1}(x') = b_i$. Furthermore, we have $h^{T'}_{4\cdot\beta+3}(x') = h' + \nabla$, where $\nabla = -1$ if $\Delta = L$ and $h' \neq 0$, $\nabla = 0$ if $\Delta = L$ and $h' = 0$, and $\nabla = 1$ if $\Delta = R$, and we have $q^{T'}_{4\cdot\beta+3}(x') = q[D,l,\Delta]$.

Because the transition function at the states $q[D,l,\Delta]$ will do nothing but move in direction Δ, and move to state $q[A,l]$, it follows that $q^{T'}_{4\cdot\alpha}(x') = q[A,l]$: already knowing that $q^T_\alpha(x) = q_l$, this fulfils the first claim of the sublemma for the case α. It also follows that that $h^{T'}_{4\cdot\alpha}(x') = h^{T'}_{4\cdot\beta}(x') + 2\cdot\nabla$, which is easily equal to $2\cdot h^T_\alpha(x)$, fulfilling the second claim of the sublemma.

Finally, the cells: the last step did not affect the content of any of the cells, so we have $c^{T'}_{i,4\cdot\alpha,h'+1}(x') = b_i$, and $c^{T'}_{i,4\cdot\alpha,h'}(x') = a_i$ for all $i \in \{1,\ldots,n\}$. For cells at position k, where $k \neq h'$ and $k \neq h'+1$, the content at α has not changed at all in any of the steps, so $c^{T'}_{i,4\cdot\alpha,k}(x') = c^{T'}_{i,4\cdot\beta,k}(x')$. However, from this it easily follows, because the original machine T replaces its content at position h with $(a_1,\ldots,a_n,b_1,\ldots,b_n)$, leaving the remaining cells all intact during the single step of computation, that the equivalence $c^{T'}_{i,4\cdot\alpha}(x') = c^T_{i,\alpha}(x) \oplus c^T_{i+n,\alpha}(x)$ indeed holds for $i \in \{1,\ldots,n\}$. In other words, the first three requirements of the proposition are fulfilled. The fourth claim is immediate from the inductive assumption and the fact that no state of the type $q[A,i]$ was visited at stage $\beta+1$, $\beta+2$, or $\beta+3$.

(3) The case where α is a limit ordinal. Here we always have that $4\cdot\alpha = \alpha$. Assume $q^T_\alpha(x) = q_k$. This means that q_k is visited cofinally often before α during the computation of T, but it follows directly, with the inductive assumption, that whenever q_k is visited at some ordinal $\beta < \alpha$, $q[A,k]$ is visited at $4\cdot\beta < \alpha$. So $q[A,k]$ is also visited cofinally often before α. Now assume towards a contradiction that another state, with a higher index, is also visited cofinally often: because of the construction, this must be some $q[A,l]$ with $l > k$. But if $q[A,l]$ is visited cofinally often by T' before α, then q_l must be visited cofinally often by T before α, as $q[A,l]$ can only be visited at ordinals of the form $4\cdot\beta$, and we would have $q^T_\alpha(x) = q_l$, a contradiction. So $q^{T'}_{4\cdot\alpha}(x') = q^{T'}_\alpha(x') = q[A,k]$. For the second claim of the sublemma, note that because α is a limit ordinal, $h^{T'}_{4\cdot\alpha}(x') = 0 = 2\cdot h^T_\alpha(x)$.

For the third claim, we have to show two things: (a) if a certain cell in one of the tapes of T changes cofinally often before α, then the corresponding cell of T' also changes cofinally often, and (b) if a cell stabilizes at either 0 or 1, the corresponding cell also stabilizes at the same value. This, however, is a simple consequence of the fact that, if a certain cell does not change during a computation of T between β and $\beta+1$, the corresponding cell also does not

change during the computation of T' between $4 \cdot \beta$ and $4 \cdot \beta + 4$; and if a cell changes once during a computation of T between β and $\beta+1$, the corresponding cell also changes exactly once during the computation of T' between $4 \cdot \beta$ and $4 \cdot \beta + 4$. The last claim is immediate from the inductive assumption and the fact that $\alpha = 4 \cdot \alpha$. □

The original lemma now follows, from the above sublemma, and the fact that, if T reaches the halting state at α, then T' reaches the halting state at $4 \cdot \alpha$. □

Lemma 2.23. *For $n \geq 1$, there is an* ITTM_1 *T with n tapes computing the operation ψ, such that for all reals $x_1, \ldots, x_n, y_1, \ldots, y_n$, $\psi(x_1 \oplus y_1, \ldots, x_n \oplus y_n) = (\{k+1 : k \in x_1\}, \ldots, \{k+1 : k \in x_n\})$, contracting all the x_i, discarding the y_i, and leaving an empty cell at the first position.*

Proof. We will sketch an algorithm that performs this computation in ω time. The idea is the following: we first move the content of the tapes at position 0 to position 1, leaving zeros at position 0; then move the content of the tapes at position 4 to position 3, leaving 0s at position 4. From this point onward, we make use of a 'gap' of 0s, delimited by a 1 at the beginning, and another one at the end. From this point onward, we go, in each iteration of the computation, to the right side of the gap, replace the 1s at the end with a 0, remember the content of the next cell, and put the 1s back two cells right of where they used to; after this, we go back to the 1s delimiting the left end of the gap, write back the remembered content, and write down the 1s that delimit the left end one cell to the right. This process will be continued until stage ω, where the tape content is identical to the desired output. We will again use signal states to recognize when we are at a limit stage. Whenever we are at a limit stage, we move to the final state and the computation finishes.

Consider a machine T where the set Q contains the following states:

- States $q[b]$, $q[c]$, $q[d]$, $q[e]$, $q[f]$, $q[g]$, $q[h]$.
- For each $(a_i, \ldots, a_n) \in \{0,1\}^n$, states $q[A, a_1, \ldots, a_n]$ and $q[I, a_1, \ldots, a_n]$.
- A signal state $q[\mathrm{sig}_G]$, an initial state q_s, and a final state q_f.

And define δ as follows:

$$\begin{gathered}
\delta(q_\mathrm{s}, (a_1, \ldots, a_n)), (q[a, x_1, \ldots, x_n], (0, \ldots, 0), R) \\
\delta(q[A, a_1, \ldots, a_n], (y_1, \ldots, y_n)) = (q[b], (x_1, \ldots, x_n), R) \\
\delta(q[B], (a_1, \ldots, a_n)) = (q[C], (a_1, \ldots, a_n), R) \\
\delta(q[C], (a_1, \ldots, a_n)) = (q[F], (1, \ldots, 1), R) \\
\delta(q[D], (a_1, \ldots, a_n)) = (q[E], (1, \ldots, 1), R) \\
\delta(q[E], (0, \ldots, 0)) = (q[E], (0, \ldots, 0), R) \\
\delta(q[E], (1, \ldots, 1)) = (q[F], (0, \ldots, 0), R) \\
\delta(q[F], (a_1, \ldots, a_n)) = (q[G], (a_1, \ldots, a_n), R) \\
\delta(q[G], (a_1, \ldots, a_n)) = (q[\mathrm{sig}_G], (1, \ldots, 1), 0), \\
\delta(q[\mathrm{sig}_n], (1, \ldots, 1)) = (q[H], (1, \ldots, 1), L)
\end{gathered}$$

$$\delta(q[H], (a_1, \ldots, a_n)) = (q[I, a_1, \ldots, a_n], (0, \ldots, 0), L)$$
$$\delta(q[I, a_1, \ldots, a_n], (0, \ldots, 0)) = (q[I, a_1, \ldots, a_n], (0, \ldots, 0), L)$$
$$\delta(q[I, a_1, \ldots, a_n], (1, \ldots, 1)) = (q[D], (a_1, \ldots, a_n), R)$$
$$\delta(q[\mathrm{sig}_g], (0, \ldots, 0)) = (q_\mathrm{f}, (0, \ldots, 0), 0)$$

Now it remains to be shown that this machine performs the desired computation. The computation can be divided in two segments: one initial segment, consisting of of the first nine stages of the computation, and a repeating segment, consisting of repeatedly visiting the states from groups D until I as well as $q[\mathrm{sig}_N]$, that will take up the rest of the computation.

For the first few stages, if $c^T_{i,0} = \langle a_0, a_1, a_2, \ldots \rangle$, the computation can be summarized in the following table. We show the stage we are at, the beginning of an arbitrary tape, the head position, as well as the stage. To make things clearer, the position of the head will also be shown in the tape using square brackets.

α	$c^T_{i,\alpha}$	h^T_α	q^T_α
0	$[a^0], a^1, a^2, a^3, a^4, a^5, a^6, \ldots$	0	q_s
1	$0, [a^1], a^2, a^3, a^4, a^5, a^6, \ldots$	1	$q[A, a^0_1, \ldots a^0_n]$
2	$0, a^0, [a^2], a^3, a^4, a^5, a^6, \ldots$	2	$q[B]$
3	$0, a^0, a^2, [a^3], a^4, a^5, a^6, \ldots$	3	$q[C]$
4	$0, a^0, a^2, 1, [a^4], a^5, a^6, \ldots$	4	$q[F]$
5	$0, a^0, a^2, 1, a^4, [a^5], a^6, \ldots$	5	$q[G]$
6	$0, a^0, a^2, 1, a^4, [1], a^6, \ldots$	5	$q[\mathrm{sig}_G]$
7	$0, a^0, a^2, 1, [a^4], 1, a^6, \ldots$	4	$q[H]$
8	$0, a^0, a^2, [1], 0, 1, a^6, \ldots$	3	$q[I, a^4_1, \ldots, a^4_n]$
9	$0, a^0, a^2, a^4, [0], 1, a^6, \ldots$	4	$q[D]$

We now go on to make the following inductive claim: consider the function f, recursively defined on all natural numbers $k > 3$, with $f(3) = 9$, and $f(n+1) = f(n)+2n+2$. We now claim that, that for each i, and each $k > 3$, if $c^T_{i,0} = x_i \oplus y_i$, then:

$$c^T_{i,f(k)} = \{m + 1 : m \in x_i \wedge m \leq k\} \cup \{2k - 1\}$$
$$\cup \{2m : m \in x_i \wedge 2m > 2k - 1\} \cup \{2m + 1 : m \in y_i \wedge 2m + 1 > 2k - 1\}$$

In other words, up to position k, $c^T_{i,f(k)}$ looks like the desired output; between positions $k + 1$ and $2k - 2$ it is empty, containing a range of $k - 2$ zeros; it contains a 1 at the position $2k-1$, and after position $2k-1$, it looks like $x_i \oplus y_i$. Furthermore, we claim that for each k, $q^T_{f(k)} = q[D]$, and $h^T_{f(k)} = k + 1$, and we claim that for all n in between $f(k)$ and $f(k + 1)$, $h^T_n > k$.

To see that the inductive claim holds for all natural numbers up to k, consider the following table showing the computation between stage $f(k)$ and $f(k+1) = f(k) + 2k + 2$. Here the tape contents between $k - 1$ and $k + 2$, and between $2k - 2$ to $2k + 1$ are shown, and we know, by the inductive assumption, that there can only be zeros between the positions $k + 2$ and $2k - 2$ of the tapes. Moreover, these zeros are left intact during this part of the computation: when T reads zeros in state $q[E]$ or any state from the $q[I, \ldots]$-group, it does not

change them, but only moves the head left or right, respectively.

α	$k-1,\ldots,k+2$	$2k-2,\ldots 2k+1$	h^T_α	q^T_α
$f(k)$	$a,b,[0],0$	$0,1,c,d$	$k+1$	$q[D]$
$f(k)+1$	$a,b,1,[0]$	$0,1,c,d$	$k+2$	$q[E]$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$f(k)+k-3$	$a,b,1,0$	$[0],1,c,d$	$2k-2$	$q[E]$
$f(k)+k-2$	$a,b,1,0$	$0,[1],c,d$	$2k-1$	$q[E]$
$f(k)+k-1$	$a,b,1,0$	$0,0,[c],d$	$2k$	$q[F]$
$f(k)+k$	$a,b,1,0$	$0,0,c,[d]$	$2k+1$	$q[G]$
$f(k)+k+1$	$a,b,1,0$	$0,0,c,[1]$	$2k+1$	$q[\mathrm{sig}_G]$
$f(k)+k+2$	$a,b,1,0$	$0,0,[c],1$	$2k$	$q[H]$
$f(k)+k+3$	$a,b,1,0$	$0,[0],0,1$	$2k-1$	$q[I,c_1,\ldots,c_n]$
$f(k)+k+4$	$a,b,1,0$	$[0],0,0,1$	$2k-2$	$q[I,c_1,\ldots,c_n]$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$f(k)+2k$	$a,b,1,[0]$	$0,0,0,1$	$k+2$	$q[I,c_1,\ldots,c_n]$
$f(k)+2k+1$	$a,b,[1],0$	$0,0,0,1$	$k+1$	$q[I,c_1,\ldots,c_n]$
$f(k)+2k+2$	$a,b,c,[0]$	$0,0,0,1$	$k+2$	$q[D]$

The changes on the tape between stage $f(k)$ and $f(k)+2k+2$ consist of the erasing of the 1s at position $2k-1$, the moving of any possible content at position $2k$ to position $k+1$, and replacing any content at $2k+1$ with 1s. From this it follows that, if we had

$$c^T_{i,f(k)} = \{m+1 : m \in x_i \wedge m \leq k\} \cup \{2k-1\}$$
$$\cup \{2m : m \in x_i \wedge 2m > 2k-1\} \cup \{2m+1 : m \in y_i \wedge 2m+1 > 2k-1\}$$

then we now also have

$$c^T_{i,f(k+1)} = \{m+1 : m \in x_i \wedge m \leq (k+1)\} \cup \{2(k+1)-1\}$$
$$\cup \{2m : m \in x_i \wedge 2m > 2(k+1)-1\} \cup \{2m+1 : m \in y_i \wedge 2m+1 > 2(k+1)-1\}$$

so the first part of the inductive claim holds for the case $k+1$. The other claims follow immediately from a look at the shown part of the computation.

It should be noted that this picture is not entirely accurate in the case of $n-3$: in this case, there is only one zero, as opposed to the two shown. The only difference here, however, that less zeros will be read, and the stages $[f(k)+1, f(k)+k-3]$ and $[f(k)+k+4, f(k)+2k]$ of the computation, where only zeros are read and the tape is moved, are skipped.

The last part of the claim that has just been inductively proven, can easily be strengthened to $h^T_n > k$ for all $n \geq f(k)$. This however implies that for every cell, there is a stage n such that the cell will never be changed after stage n before ω. But now it follows that $c^T_{i,\omega} = \{m+1 : m \in c^i\}$, as desired.

Finally, at stage ω, we will again be in state $q[\mathrm{sig}_G]$, but this time read a 0, and move to the final state. □

Lemma 2.24. *For all $n \geq 1$, the class of operations from n-tape inputs to n-tape outputs by n-tape* ITTM_1 *machines is closed under composition, i.e. if there is a n-tape machine S computing ψ_S, and a n-tape machine R computing ψ_R, there is a n-tape machine T computing ψ_T, such that $\psi_T(x_1,\ldots,x_n) = \psi_S(\psi_R(x_1,\ldots,x_n))$ for all $(x_1,\ldots,x_n) \in \mathbb{R}^n$.*

Proof. Say R contains m states named $q_1, \ldots, q_m$, and S contains p states named $q_1, \ldots, q_p$. To compose these two operations, we need to (a) start out with a simulation of R, and then, when we have reached the state corresponding to the final state of R, (b) move the head back to position 0, and (c) continue with a simulation of S. The 'move left' part might seem straightforward, but we also need to know when we have finished moving left, for which we, once again, need a construction with signal states. Consider the machine T with the following states:

- States $q[R, i]$ for each $i \in \{1, \ldots, m\}$ corresponding to the states of R.
- States $q[A, i]$ for $i \in \{1, \ldots, 4\}$ for the moving left.
- States $q[S, i]$ for each $i \in \{1, \ldots, p\}$ corresponding to the states of S.

If q_j is the initial state of R, then $q[R, j]$ will be the initial state of T, and if q_k is the final state of S, then $q[S, k]$ will be the final state of T. We furthermore make sure that the states in the R-group have the lowest identifying numbers; the states in the A group have higher identifying numbers, and the states in the S group have the highest identifying numbers. In each group, furthermore the internal order will be so that $q[X, i] \leq q[X, j]$ iff $i \leq j$.

We now define the function δ as follows:

- For all $(a_i, \ldots, a_n), (b_i, \ldots, b_n) \in \{0, 1\}^n$, all $j, k \in \{1, \ldots, m\}$, and all $\Delta \in \{L, R\}$, if

  $$\delta_R(q_j, (a_i, \ldots, a_n)) = (q_k, (b_1, \ldots, b_n), \Delta)$$

  we set

  $$\delta_T(q[R, j], (a_i, \ldots, a_n)) = (q[R, k], (b_1, \ldots, b_n), \Delta)$$

- For all $(a_i, \ldots, a_n), (b_i, \ldots, b_n) \in \{0, 1\}^n$, all $j, k \in \{1, \ldots, p\}$, and all $\Delta \in \{L, R\}$, if

  $$\delta_S(q_j, (a_i, \ldots, a_n)) = (q_k, (b_1, \ldots, b_n), \Delta)$$

  we set

  $$\delta_T(q[S, j], (a_i, \ldots, a_n)) = (q[S, k], (b_1, \ldots, b_n), \Delta)$$

- If q_k is the final state of R, we set, for all $(a_i, \ldots, a_n) \in \{0, 1\}^n$:

  $$\delta_T(q[R, k], (a_i, \ldots, a_n)) = (q[A, 1], (a_i, \ldots, a_n), 0)$$

- For all $(a_i, \ldots, a_n) \in \{0, 1\}^n$, we set:

  $$\begin{aligned}
  \delta_T(q[A, 1], (1, a_2, \ldots, a_n)) &= (q[A, 4], (0, a_2, \ldots, a_n), 0) \\
  \delta_T(q[A, 1], (0, a_2, \ldots, a_n)) &= (q[A, 2], (1, a_2, \ldots, a_n), 0) \\
  \delta_T(q[A, 2], (1, a_2, \ldots, a_n)) &= (q[A, 3], (0, a_2, \ldots, a_n), 0) \\
  \delta_T(q[A, 3], (0, a_2, \ldots, a_n)) &= (q[A, 1], (0, a_2, \ldots, a_n), L) \\
  \delta_T(q[A, 4], (0, a_2, \ldots, a_n)) &= (q[A, 1], (1, a_2, \ldots, a_n), L)
  \end{aligned}$$

- If q_k is the initial state of S, we set, for all $(a_i, \ldots, a_n) \in \{0,1\}^n$:

$$\delta_T(q[A,3],(1,a_2,\ldots,a_n)) = (q[S,k],(0,a_2,\ldots,a_n),0)$$
$$\delta_T(q[A,4],(1,a_2,\ldots,a_n)) = (q[S,k],(1,a_2,\ldots,a_n),0)$$

It is easy to verify by induction that, if q_k is the final state from R, and we have $q^R_\alpha(x_1,\ldots,x_n) = q_k$ and $c^R_{i,\alpha}(x_1,\ldots,x_n) = y_i$ for $i \in \{1,\ldots,n\}$, then we also have $q^T_\alpha(x_1,\ldots,x_n) = q[R,k]$, $c^T_{i,\alpha}(x_1,\ldots,x_n) = y_i$ for $i \in \{1,\ldots,n\}$.

It is also easy to verify, again by induction, that if q_j is the initial state from S, and q_k is the final state from S, and we have for some ordinal β that $q^T_\beta(x_1,\ldots,x_n) = q[S,j]$, for all i, $c^T_{i,\beta}(x_1,\ldots,x_n) = y_i$, and $h^T_\beta(x_1,\ldots,x_n) = 0$; and, for some other ordinal η $q^S_\eta(y_1,\ldots,y_n) = q_k$ and $c^S_{i,\eta}(y_1,\ldots,y_n) = z_i$ for all $i \in \{1,\ldots,n\}$, then we also have $q^T_{\beta+\eta}(x_1,\ldots,x_n) = q[S,k]$, and $c^T_{i,\beta+\eta}(x_1,\ldots,x_n) = z_i$ for $i \in \{1,\ldots,n\}$.

In order to complete the proof, we only need to show that, if $q^T_\alpha(x_0,\ldots,x_n) = q[R,k]$ where $q[R,k]$ corresponds to the final state of R, there is some other ordinal β such that $q^T_\beta(x_0,\ldots,x_n) = q[S,j]$, where $q[S,j]$ corresponds to the initial state of S, where the contents of the tapes is identical at stages α and β, and the head position at β is zero. An inspection of the definition of δ_T however reveals that $\alpha+\omega+1$ is this β. At stage α, we move from state $q[R,k]$ to state $q[A,1]$. If, in state $q[A,1]$, we read a 1 on the first tape, we go to $q[A,4]$, replacing the 1 by a 0, and then back to $q[A,1]$, replacing the 0 again by a 1 (thus restoring the tape to the earlier position), and moving the head left. If, in state $q[A,1]$, we read a 0 on the first tape, we go to $q[A,2]$, replacing the 0 by a 1, then to $q[A,3]$, replacing the 1 by a 0 again, and then back to $q[A,1]$ while moving the head left (again restoring the tape to the earlier position). When we are at position 0, we cannot move further left, thus this sequence happens finally often, and the content of the first cell of the first tape is switched between 0 and 1 cofinally often before $\alpha+\omega$. However, if the original content of that cell is a 0, then $q[A,3]$ is visited cofinally often, whereas, if it is a 1, then $q[A,4]$ is visited cofinally often. So, at stage $\alpha+\omega$, we find ourselves either in $q[A,3]$ or in $q[A,4]$, reading a 1 on the first cell of the first tape. If we are in $q[A,3]$, we know this 'should' be a zero, and if we are in $q[A,4]$, we know this 'should' be a 1. We restore the content of the tape which now again has the same contents that it had at stage α, and go on to $q[S,j]$ to start the last part of the computation.

From this, it follows that ψ_T is indeed the desired composition $\psi_S \circ \psi_R$. □

Proposition 2.25. *For any $n \geq 2$, and any f, if f is computable by an* ITTM_1 *T with $2n$ tapes, then f is computable by an* ITTM_1 *T' with n tapes.*

Proof. We do this by composing the functions that have been shown to exist in the Lemmata 2.19-2.24. For each $i \in \{19,\ldots,23\}$, let ϕ_i be the function that has been shown to exist in Lemma 2.i with respect to n (and, in the case of Lemma 2.23, with respect to T). Defining $\phi_a := \phi_{21} \circ \phi_{21} \circ \phi_{20} \circ \phi_{19}$ and $\phi_b := \phi_{21} \circ \phi_{23}$, we observe that:

$$\phi_a(x_1,\ldots,x_n) = (\{2k : k \in x_1\},\ldots,\{2k : k \in x_n\}) = (x_1 \oplus \emptyset,\ldots,x_n \oplus \emptyset)$$

and that

$$\phi_b(x_1 \oplus y_1,\ldots,x_n \oplus y_n) = (x_1,\ldots,x_n)$$

It is now easy to see that $\phi_b(\phi_{22}(\phi_a(x_1, \ldots, x_n)))$ is the same function as ϕ_T. By repeated application of Lemma 2.24, it follows that there is an ITTM$_1$ T' with n tapes computing this function. □

Interestingly, it turns out that ITTM$_1$ machines with only one tape *do* have the full power of ITTM$_1$ machines with more tapes. The main reason why the earlier ITTM$_0$ machines with one tape were strictly less powerful than ITTM$_0$ machines with multiple tapes, was that the one-tape machines were not closed under composition. However, Lemma 2.24 also works in the case where $n = 1$, giving the following result:

Proposition 2.26. *For any function f, if f is computable by an* ITTM$_1$ *T with 2 tapes, then f is computable by an* ITTM$_1$ *T' with 1 tape.*

Proof. Because none of the Lemmata 2.19-2.24 required the number of tapes to be larger than 1, the proof of Proposition 2.25 carries over almost directly to the one-tape case. The only possible complication here, is that in the simulated machine T, the output appears on the second tape, so we have to make sure that, if $x \oplus y$ appears on the only tape of our one-tape machine after the simulation of T, this is transformed into y, rather than into x. With all ϕ_i and ϕ_a as in the Proposition 2.25, and defining $\phi_c := \phi_{21} \circ \phi_{21} \circ \phi_{23} \circ \phi_{19}$, we observe

$$\phi_c(x_1 \oplus y_1) = (y_1)$$

and it is now easy to see that $\phi_c(\phi_{12}(\phi_a(x_1, \ldots, x_n)))$ is the same function as ϕ_T. Again, this function is computable by a one-tape machine by repeatedly applying Lemma 2.24. □

Of course, machines with more tapes can, as always, 'simulate' machines with less tapes:

Proposition 2.27. *For any $n, m \geq 1$, with $n > m$, if f is computable by an* ITTM$_1$ *with m tapes, then f is computable by an* ITTM$_1$ *with n tapes.*

Proof. This is extremely simple: given a definition of the machine with m tapes, we can convert it to a definition of a machine with n tapes computing the same function, by simply ignoring the extra tapes. □

Theorem 2.28. *For any $n, m \geq 1$, and for any function f, f is computable by an* ITTM$_1$ *with n tapes if and only if it is computable by an* ITTM$_1$ *with m tapes.*

Proof. We can assume, without loss of generality, that $n > m$. That we can simulate a machine with m tapes by a machine with n tapes, now is immediate from Proposition 2.27. For the other direction, note that by repeatedly applying Proposition 2.25 (and, if $m = 1$, applying Proposition 2.26 once) we can conclude that we can simulate any machine with $m \cdot 2^p$ tapes for some p such that $m \cdot 2^p > n$, by a machine with m tapes, and Proposition 2.27 gives us that we can simulate any machine with n tapes by a machine with $m \cdot 2^p$ tapes. Thus, we can simulate any machine with n tapes by a machine with m tapes. □

Finally, we again note that we may always add any finite number of scratch cells at will:

Proposition 2.29. *For any $n \geq 1$, and for any $m \geq 1$, the set of functions computable by* ITTM_1 *machines with n tapes and m scratch cells is equivalent to the set of functions computable by* ITTM_1 *machines with n tapes without any scratch cells.*

Proof. The proof of Proposition 2.10 generalizes to the ITTM_1 case with minimal modifications. □

2.5 Equivalence of ITTM_0 and ITTM_1

We now return to the question, whether it is possible to simulate an ITTM_1 by an ITTM_0, and vice versa. It turns out that usage of flags and extra states indeed allows us to do so, in both directions:

Proposition 2.30. *For any function f, if f is computable by an* ITTM_1*, then f is computable by an* ITTM_0*.*

Proof. Given a certain ITTM_1 T (with three tapes), consisting of n states $q_1, \ldots, q_n$, we can simulate its computation by an ITTM_0 T' with three tapes and n flags and stay option in the following way. The main idea of the simulation will be to use flags to record how often each of the states is visited and, when we find ourselves in the limit state, use the contents of the flags to determine which state to move to. The details of the process will be the following:

Say, the original machine T has states $q_1, \ldots, q_n$. The new machine T' will have these states, as well as n new states $q_{n+1}, \ldots q_{2n}$ and a limit state $q_{\mathrm{lim}} = q_{2n+1}$. For each transition of some q_j to q_k in the original δ, we now first go from q_j to q_{n+k} while carrying out the essential modifications on the tape and moving left or right, as well as writing a 1 on the $j+1$th flag. Then, at position q_{n+k}, we do nothing except for erasing the 1 again and going ahead to position q_k. So, each time a state is visited, the flag corresponding to that state is toggled to 1 and back to 0 again. When we are in the limit state, we look at all the flags, and move to the state corresponding to the highest flag that is set to 1, while setting all of the flags back to zero. It should be clear that this machine T' computes the exact same function as the original machine T. □

Proposition 2.31. *For any function f, if f is computable by an* ITTM_0*, then f is computable by an* ITTM_1*.*

Proof. The other way around, if we are given a certain ITTM_0 T with states $q_1, \ldots, q_n$, again with three tapes, we can simulate it by an ITTM_1 T' with three tapes and one flag and stay option. This can be done in the following way: in addition to the original states $q_1, \ldots, q_n$, the new machine T' will have n additional states, $q_{n+1}, \ldots, q_{2n}$. For any transition from q_j to q_k in the original machine T, we now first move from q_j to q_{n+k}, performing all the important modifications on the tapes and head, and making sure a 0 is written on the flag. At q_{n+k}, we will do nothing besides again writing a 1 on the flag, and going ahead to stage q_k.

Because every time one of the original states is visited at a non-limit stage, it is preceded by the visiting of one of these extra states, and because the extra stages are numbered higher than the original states, we will, at a limit stage, always find ourselves in one of these new states. And, because the additional

states are visited cofinally often, at limit stages we will will read a 1 on the extra flag, signaling that we are in a limit state. By reading a 1 on this flag in one of the new states, we recognize that we are in fact in a limit state, and we now go on to the special limit state. □

It should be noted that the ITTM_1 just defined is actually able to 'see' whenever it is at a limit stage of the computation. This fact will be used in a few later theorems!

All in all, it turns out that *almost* all of the types of infinite time Turing machines are equivalent with regards to the class of functions they can compute: the only exception we have, are ITTM_0 machines with only one tape and no additional flags.

Still, this will not be our last word on the equivalence between different types of Hamkins-Kidder machines: after we have defined the important time and space complexity classes in chapter 4, we still need to find out whether those are invariant between the different types and subtypes of machines defined in this chapter.

Chapter 3

Clockability and writability

Two notions that will play an important role in the remainder of this thesis, are the notion of clockability of ordinals, and the notion of writability of reals as well as of ordinals. In [HaLe], a variety of theorems about clockable and writable ordinals are proven, of which we will mention the most relevant ones here. It should be noted that, in this chapter, we only consider ITTM_0 machines with three tapes—in all likelihood, however, invariance of writability, clockability, etc. between most of the variations from chapter 2 can be easily, but tediously, proven.

3.1 Definitions

Definition 3.1. We call an ordinal α clockable if there is an infinite time Turing machine T, such that $q^T_{\alpha+1}(\emptyset) = q_{\mathrm{f}}$ or, in other words, that T reaches the halt state at time $\alpha + 1$. Likewise, we call an ordinal α x-clockable if there is an infinite time Turing machine T that reaches the halting state q_{f} at time $\alpha+1$ on input x. We call the supremum of all clockable ordinals γ, and the supremum of all x-clockable ordinals γ^x.

One may wonder why we have used the ordinal $\alpha + 1$ here, instead of the more obvious α. The prime reason for this is convention: this was the original definition in [HaLe], and a lot of the results in earlier papers about clockable ordinals would need to be modified if we left out the $+1$ part. Besides this, an inspection of the definitions in chapter 2 should make it obvious that it is impossible that $q^T_\alpha(x) = q_{\mathrm{f}}$ for any x and any limit ordinal α. In the case of ITTM_0 machines, this is because we have $q_{\mathrm{f}} \neq q_{\mathrm{lim}}$. In the case of ITTM_1 machines, it is likewise impossible to first reach q_{f} at a limit ordinal stage, due to the fact that, if we are in any state at a limit ordinal stage, this means that this state has been visited before cofinally often.

Before going on defining the notion of writable reals and ordinals, let us make precise what we mean when we say that a real 'codes an' ordinal:

Definition 3.2. Using an arbitrary pairing function $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, we can regard any real a as a relation R on the natural numbers, by considering it as a subset of $\mathbb{N} \times \mathbb{N}$. We say that a real a codes an ordinal α if there is an isomorphism between $(\mathrm{Field}(R), R)$ and $(\alpha, \in)$.

Definition 3.3. We call a real v writable if there is an infinite time Turing machine T, such that $\phi_T(\emptyset) = v$. We call an ordinal α writable if there is a writable real x coding α. We call a real v x-writable, for another real x, if there is an infinite time Turing machine T such that $\phi_T(x) = v$. We call an ordinal α x-writable, if there is an x-writable real coding α. We call the supremum of all writable ordinals λ, and the supremum of all x-writable ordinals λ^x.

Definition 3.4. We call a real v eventually writable if it is writable, or if there is an infinite time Turing machine T, such that $\phi_T(\emptyset) \uparrow$, and there is an ordinal α, such that $c^T_{2,\beta}(\emptyset) = v$ for all $\beta > \alpha$. In this case, T at some stage writes v on the output tape, never to change it afterwards, but without ever halting. Likewise, we can again define eventually writable ordinals, eventually x-writable reals, and eventually x-writable ordinals, analogous to Definition 3.3. We call ζ the supremum of all eventually writable ordinals, and ζ^x the supremum of all eventually x-writable ordinals.

Definition 3.5. We call a real v accidentally writable if there is an infinite time Turing machine T, some $i \in \{1, 2, 3\}$, and some ordinal α, such that $c^T_{i,\alpha}(\emptyset) = v$. Likewise, we can define accidentally writable ordinals, and accidentally x-writable reals and ordinals, again analogous to Definition 3.3. We call Σ the supremum of all accidentally writable ordinals, and Σ^x the supremum of all accidentally x-writable ordinals.

Definition 3.6. For any $i \in \{1, 2, 3\}$, and any $n \in \mathbb{N}$, we say that cell n of tape i stabilizes before α on a computation by T from x, if there is an ordinal $\beta < \alpha$, such that $c^T_{i,n,\beta}(x) = c^T_{i,n,\alpha}(x)$, and $c^T_{i,n,\beta}(x) = c^T_{i,n,\eta}(x)$ for all η with $\beta < \eta < \alpha$. We say that an cell has stabilized *at* an ordinal β, before α, if $c^T_{i,n,\beta}(x) = c^T_{i,n,\alpha}(x)$, and $c^T_{i,n,\beta}(x) = c^T_{i,n,\eta}(x)$ for all η with $\beta < \eta < \alpha$. In the latter case, the specification 'before α' may be left out if it is clear from the context.

3.2 Results

It easily follows from the definitions that, for any $x \in \mathbb{R}$, λ^x and ζ^x are countable. After all, reals can only code countable ordinals, there are only countably many infinite time Turing machines, and each of them can only write, or eventually write, one ordinal at most starting from x. Hence λ^x and ζ^x are unions of countably many countable ordinals, and are thus also countable. In the case of γ^x and Σ^x, this is less obvious: on a first glance, nothing seems to exclude the possibility that very large, uncountable, ordinals may be clockable, or that a single machine T can end up writing a code for every single countable ordinal accidentally, which would imply that $\Sigma^x = \omega_1$. However, it has turned out that both Σ^x and γ^x are in fact countable. We will soon give with a reproduction of [HaLe, Theorem 1.1].

We start by giving the following lemma:

Lemma 3.7. *If α and β are ordinals, and the snapshot at α is identical to the snapshot at β, then, for all ordinals η, the snapshot at $\alpha + \eta$ is identical to the snapshot at $\beta + \eta$.*

Proof. This can easily be proven using ordinal induction on η. □

Proposition 3.8. *For all* ITTM_0*s* T *and all reals* x*, there are countable ordinals* α *and* β*, with* $\alpha < \beta$*, such that the snapshots at* α*,* β*, and at* ω_1 *are identical, and such that for any ordinal* $\mu > \beta$*, there is an ordinal* δ *with* $\alpha \leq \delta < \beta$*, such that the snapshots at* μ *and at* δ *are identical.*

Proof. Taking any ITTM T and any real x, we define the two following sets, with respect to the computation by T from x:

$$A = \{(n, i) : \text{cell } n \text{ of tape } i \text{ stabilizes before } \omega_1\}$$

$$B = \{(n, i) : \text{cell } n \text{ of tape } i \text{ does not stabilize before } \omega_1\}$$

We furthermore define the function $f : A \to \omega_1$ that assigns to each $(n, i) \in A$ the least ordinal η such that cell n of tape i is stabilized at η. Looking at the range of f, we see a countable set of countable ordinals, the supremum of which is also a countable ordinal. We set $\theta = \sup f[A]$, and now know that $\theta < \omega_1$, and that all the cells that stabilize before ω_1 are already stabilized at θ.

For each $(n, i) \in B$, we know that the nth cell of tape i does not stabilize before ω_1, so it has to change at some point between θ and ω_1. We thus can define the function $g_0 : B \to \omega_1$ that assigns to each $(n, i) \in B$ the least ordinal η larger than θ such that $c^T_{i,n,\eta}(x) \neq c^T_{i,n,\theta}(x)$. Setting $\alpha_0 = \sup g_0[B]$, we obtain a countable ordinal α_0 such that every cell that has not stabilized at θ has changed at least once (but possibly more often) between θ and α_0. We can repeat this process, and for each α_j define α_{j+1} as the least ordinal larger than α, such that each cell in B has changed at least once between α_j and α_{j+1}. We now set $\alpha = \sup\{\alpha_i : i \in \mathbb{N}\}$, obtaining another countable ordinal, strictly larger than all α_i. The ordinal α must be a limit ordinal, as it is a supremum of an infinite, increasing, sequence of ordinals, so we get $q^T_\alpha(x) = q_{\lim} = q^T_{\omega_1}(x)$, and $h^T_\alpha(x) = 0 = h^T_{\omega_1}(x)$. For all $(n, i) \in A$, we must have that $c^T_{n,i,\alpha}(x) = c^T_{n,i,\omega_1}(x)$, because these cells are all stabilized at θ. For all $(n, i) \in B$, it is immediate that a cell n at tape i cannot possibly stabilize before α, because it has to change at least between each α_j and each α_{j+1}. It thus follows that $c^T_{n,i,\beta}(x) = 1 = c^T_{n,i,\omega_1}(x)$, and the result $c^T_{i,\alpha}(x) = c^T_{i,\omega_1}(x)$ follows.

Starting from the ordinal α that has been obtained, we can repeat the above process again, to find another increasing sequence of ordinals $\beta_i : i \in \mathbb{N}$, where every cell $(n, i) \in B$ changes at least once between α and β_0, and between β_i and β_{i+1} for all $i \in \mathbb{N}$, and eventually another ordinal $\beta = \sup\{\beta_i : i \in \mathbb{N}\}$ at which the snapshot is again identical to that at ω_1, and now necessarily also identical to that at α. This concludes the proof of the first part of the proposition.

For the second claim of the proposition, assume towards a contradiction that it does not hold, and let $\mu > \beta$ be the smallest ordinal for which it fails. We now define C as the set of all ordinals smaller than μ at which the snapshot is equal to that at ω_1, and η as the supremum of this set. It follows that the snapshot at η itself is equal to that at ω_1: all of the cells in A must have the same value as at ω_1, because $\eta \geq \beta$, and all of the cells in B must have the value 1, like at stage ω_1, because either $\eta \in C$, or there are cofinally many ordinals θ below η such that the snapshot at θ is identical to that at ω. Now let $\kappa = \mu - \eta$. If $\kappa \geq \beta - \alpha$, it follows that $\eta + (\beta - \alpha) \leq \mu$, and by Lemma 3.7 it would then follow that the snapshot at $\eta + (\beta - \alpha)$ is identical to that at $\omega_1 + (\beta - \alpha)$, and hence to $\beta = \alpha + (\beta - \alpha)$, and hence to ω_1, a contradiction. But if $\kappa < (\beta - \alpha)$,

it would follow from the same lemma that the snapshot at stage $\mu = \eta + \kappa$ is identical to that at $\alpha + \kappa$, which is below β, so again a contradiction. It thus follows that the second claim must hold for all ordinals larger than β. □

Proposition 3.9. *For all reals x, Σ^x is countable.*

Proof. For all machines T and all reals x, there is, as a result of Proposition 3.8, a countable ordinal β, such that all reals that are accidentally written by T from x must be written at some point before β. So any machine T can only accidentally write countably many different ordinals, and, because there are only countably many different ITTMs, it follows that Σ^x is a countable union of countable sets of countable ordinals, and hence countable. □

Proposition 3.10. *For all reals x, γ^x is countable.*

Proof. If α is an x-clockable ordinal, then α must be countable as a result of Proposition 3.8. Because there are only countably many machines T, there can only be countably many x-clockable ordinals, so γ^x is a countable union of countable ordinals, and hence countable. □

It is immediate from the definitions that every writable ordinal is eventually writable, and every eventually writable ordinal is accidentally writable. So at the very least, we know that $\lambda^x \leq \zeta^x \leq \Sigma^x$. It has turned out that this result can be strengthened to $\lambda^x < \zeta^x < \Sigma^x$:

Proposition 3.11. *For any real x, λ^x is eventually x-writable.*

Proof. For any x, consider the relation $\lhd$, where $p \lhd q$ if $\phi_p(x)$ and $\phi_q(x)$ both are finishing computations that output the code for an ordinal, and $\phi_p(x)$ codes a smaller ordinal than $\phi_q(x)$. It is clear that this relation has the same order type as λ^x, and thus is a code for λ^x. However, we can eventually write this relation, with an infinite time Turing machine T that simultaneously simulates the computation of all other infinite time Turing machines on input x, using the first tape for this simulation. Whenever one of the ω many simulated computations, say, that of ϕ_p ends, we compare it to all the computations ϕ_q that finished earlier on: for any of these q that output a real coding a smaller ordinal than the ordinal coded by p, we write a 1 at position $\langle q, p\rangle$ of the output tape, and for any of these q that output a real coding a larger ordinal than p, we write a 1 at position $\langle p, q\rangle$ of the output tape. Onward from the point where the simultaneous computation of all infinite time Turing machines has simulated every stage before λ^x of every machine, we can be sure that none of the simulated computations will ever finish at any later point, so the output tape stays intact from this point onward. But exactly at this point the relation $p \lhd q$ is written on the output tape: it follows that λ^x is eventually writable. □

Proposition 3.12. *For any x, the class of accidentally x-writable ordinals is closed downward.*

Proof. Consider a three-tape machine T that accidentally writes some ordinal α from input x. We can now construct a machine T' that first stretches the input x such that, after the stretching operation, we have $\{3 \cdot k : k \in x\}$ on the tape, and then simulates the operation of T' using only the first tape. After each step, we check for each of the three simulated tapes if it contains a real

x coding an ordinal α. If it does, we write, in turn, every restriction of x, and hence, every ordinal $\beta < \alpha$ to one of the two other tapes. But now T' clearly accidentally writes every ordinal $\beta < \alpha$, proving that the class of accidentally writable ordinals is closed downward. □

Proposition 3.13. *For any real x, ζ^x is accidentally x-writable.*

Proof. Consider a computation that, stage by stage, simulates the computation of all computations on input x. At any stage, after that step of the simulation has finished, we check which of the snapshots code ordinals, and write a code for the union of those ordinals to the output tape. Because every non-halting computation will eventually repeat at some countable stage, all eventually writable ordinals are certainly written down at the supremum of these stages. So, at this stage, some ordinal larger than ζ^x is written to the tape. Because the class of accidentally writable ordinals is downward closed, it follows that ζ^x also has to be accidentally writable. □

We can now establish the strict inequality $\lambda^x < \zeta^x < \Sigma^x$:

Proposition 3.14. *For any real x, $\lambda^x < \zeta^x < \Sigma^x$.*

Proof. From any infinite time Turing machine T, we can construct a new infinite time Turing machine T', simulating the computation of T, but simulating the output tape of T on another, non-output tape, and which, after each computation step of the simulation, checks if one of the simulated tapes contains a real coding an ordinal α and, if it does, writes a real coding $\alpha+1$ on the output tape, except in the case where a real coding $\alpha+1$ is already on the output tape, where the output tape is simply left intact. It follows that, if T eventually writes α, T' eventually writes $\alpha+1$, and if T accidentally writes α, T' accidentally writes $\alpha+1$. So any successor of an eventually or accidentally x-writable ordinal is also eventually or accidentally x-writable. Hence λ^x+1 is eventually x-writable, but it cannot be x-writable as λ^x is the supremum of all x-writable ordinals, and thus $\lambda^x \leq \zeta^x$ can be strengthened to $\lambda^x < \zeta^x$. Likewise, from the fact that $\zeta^x + 1$ is accidentally x-writable, it follows that $\zeta^x < \Sigma^x$. □

Proposition 3.15. *For any real x, we have $\lambda^x \leq \gamma^x$ and, in particular, we have $\lambda \leq \gamma$.*

Proof. This is a part of [HaLe, Order-type Theorem 3.8]. Say, α is an x-writable ordinal that is coded by the x-writable real y, and n is a natural number. Now consider an ITTM$_0$ $T_{y,n}$ that, on input x, first performs the algorithm that writes y on the third tape, and then, in exactly ω steps, deletes the input tape while writing $\{n\}$ on it. After this, it one by one erases the least element from the field of y, until the point where it finds that n is the least element left on it, at which it halts directly.

If $\beta < \alpha$, there is a unique ordinal n_β such that the erased part of y up to n_β has the same order type as β. Now assume that $\beta < \eta < \alpha$, and consider the infinite time Turing machines T_{y,n_β} and T_{y,n_η}. It is easy to see that the computation of T_{y,n_η} from x takes strictly longer than the computation of T_{y,n_β} from x: because the order type of the erased part up to n_β is smaller than the order type of the erased part up to n_η, T_{y,n_η} will at some point have n_β as least number on the tape, and continue the computation, whereas T_{y,n_η} will halt at the same stage.

Now let f be the function that maps every ordinal $\beta < \alpha$ to the time it takes for a computation by T_{y,n_β} to halt from input x. We now know that if $\beta < \eta$, we will also have $f(\beta) < f(\eta)$, with $f(\beta)$ and $f(\eta)$ both being x-clockable ordinals. This in turn implies that there are at least α many x-clockable ordinals, and hence that $\alpha \leq \gamma^x$. Because this argument works for every writable ordinal α, it can be concluded that $\lambda^x \leq \gamma^x$. □

A major question that was unsolved in [HaLe], was the question whether every clockable ordinal was writable or, in other words, whether γ was equal to λ. It was proven later that this is indeed the case. To show this, we first need the following theorem, which is an adaptation of [We, Corollary 2.1]:

Proposition 3.16. *For all reals x, and for any computation by an infinite time Turing machine T from x that does not halt by time Σ^x, the snapshot at time ζ^x is the same as that at time Σ^x.*

Proof. In order to prove this lemma, we will show two things:

1. If a cell has, during the computation of a Hamkins-Kidder machine T, at time Σ^x, stabilized at either value 0 or 1, the point of stabilization must be before time ζ^x.

2. If a cell is, during the computation of a Hamkins-Kidder machine T, not stabilized at time Σ^x, and thus has the value 1 as a result of the lim sup operation, it cannot possibly be stabilized at time ζ^x either, and thus must have the value 1 at time ζ^x, too.

Because, at time Σ^x, all cells are either stabilized, in which case the first claim applies, or not stabilized, in which case the second claim applies, the two claims combined imply that the snapshot at Σ^x is exactly that at ζ^x.

To see that the first claim holds, we assume that a position n on a tape has stabilized at time Σ^x. This means that there must be some ordinal $\delta < \Sigma^x$, such that the cell at n never changes anymore after stage δ. We will provide an algorithm that eventually writes δ, which proves that δ is eventually writable from x or, in other words, $\delta < \zeta^x$. Consider an algorithm that performs a simultaneous computation of all Hamkins-Kidder machines starting from the input x. At every stage of the computation of this algorithm, we obtain a real, for which we can test if it codes an ordinal. If we continue this process, we will eventually exhaust all ordinals accidentally writable from x or, in other words, all ordinals up to Σ^x. Whenever this algorithm provides us with an ordinal α, we simulate the computation of the machine T up to the point of this ordinal, and check if T is stabilized by time α. If it has, we look for the exact point of stabilization β. If there is no ordinal written on the output tape, we write β on the output tape; and if there is an ordinal μ on the output tape, we check if $\beta > \mu$. If it is, we write β on the output tape, and if it is not, we keep the output tape as it is. Because eventually we will find some accidentally writable ordinal that is larger than δ, at some point the true stabilizing point δ will be written on the tape; and, because position n of the tape never changes afterwards, there can be no (temporary) stabilizing point larger than the true stabilizing point, so δ will never be overwritten. It thus follows that δ is eventually writable, and hence that $\delta < \zeta^x$.

To see that the second claim holds, we assume that a position n on a tape is not stabilized by time Σ^x. As of yet, there still is a possibility that this position is temporarily stabilized at an earlier stage, such as ζ^x, and destabilizes later on. To exclude this possibility, we will start by assuming towards a contradiction that position n is stabilized by time ζ^x. In this case, there must be a point of stabilization $\delta < \zeta^x$, which is eventually writable by some machine S with input x because it is below ζ^x. Consider a machine S which eventually writes the ordinal δ starting from input x. Again, we regard this machine as providing a stream of reals, some of which may code ordinals, this time noting that the stream will eventually stop, with the ordinal δ being the final ordinal. Each time we obtain a new ordinal α from the stream, we will simulate the machine T, looking for the smallest stage β where the cell at n changes after this point α. This β must always exist, because α is accidentally writable, and we know that cell n is not stabilized at the supremum of accidentally writable ordinals Σ^x. Each time we obtain such an ordinal β, we write it to the output tape. Eventually, when the ordinal δ is passed on as the final ordinal of the stream, the first ordinal μ where n changes after δ will be written to the output tape. Because of the assumption that δ is the point of stabilization before ζ^x, however, it must follow that $\mu > \zeta^x$. This, however, is contradictory, because we have an algorithm that eventually writes μ, and ζ^x is the supremum of eventually writable ordinals from x. So the assumption that n is stabilized by time ζ^x must be false. Therefore, if a cell is not stabilized at stage Σ^x (and, because of that, has value 1 at that stage), it cannot be possibly stabilized at stage ζ^x either (and, because of that, also has value 1 at stage ζ^x). □

Right now, we still need one more auxiliary lemma to enable us to prove the equality of λ and γ:

Proposition 3.17. *If we have $\gamma^x < \Sigma^x$, then we also have $\lambda^x = \gamma^x$.*

Proof. Assume, towards a contradiction, that γ^x is larger than λ^x but smaller than Σ^x. Then there has to be a x-clockable ordinal β in between λ^x and Σ^x, which must be witnessed by some infinite time Turing machine T that reaches the halt state at stage $\beta + 1$ from input x. Now consider the algorithm that simultaneously simulates the computation of all other infinite time Turing machines; at every stage of this simulation, we obtain a real x, and check if it codes an ordinal. If x codes an ordinal η, we write a copy of x to the scratch tape, simulate the computation of T from x, and at every step of this second simulation, we erase the least element of what is left of the real x. Eventually, we either end up with an empty field, (if $\eta < \beta + 1$), in which case we go on with the original simulation, and continue the search for new ordinals; or the simulation of T finishes before x is emptied (if $\beta + 1 \leq \eta$). In the latter case, we continue by writing η on the output tape, and let the computation halt.

Because the first simulation will eventually generate a code for every accidentally x writable ordinal, the second case will always occur at some point, and some η larger than β will be written to the output tape. This however contradicts the assumption that $\beta \geq \lambda^x$, so $\lambda^x = \gamma^x$ follows from $\gamma^x < \Sigma^x$. □

Now we can show that, indeed, the suprema of the clockable and writable ordinals are identical:

Proposition 3.18. *For any x, we have $\gamma^x < \Sigma^x$ and hence, by Proposition 3.17, $\lambda^x = \gamma^x$. In particular, $\lambda = \gamma$ is true.*

Proof. Assume towards a contradiction that $\gamma^x \geq \Sigma^x$. In this case, there has to be an clockable ordinal that is not accidentally writable. In Proposition 3.16 it was shown that the snapshots at stage ζ^x and Σ^x were equal, and by the assumption that $\gamma^x \geq \Sigma^x$, there has to be a computation by an ITTM_0 T from x that terminates after Σ^x. As a result, there has to be a least ordinal $\alpha > \Sigma^x$, such that the snapshot at stage α, in the computation by T from x, does not occur between ζ^x and Σ^x. We will now show that there can be no such ordinal α, so $\gamma^x < \Sigma^x$ has to be true. Let us call the repeating snapshot that occurs at both ζ^x and Σ^x R.

It follows from Lemma 3.7 that if R occurs at some ordinal θ, R will also occur at $\theta + (\Sigma^x - \zeta^x)$. Also, if $\beta < \alpha$ is a limit of a set A of ordinals where R occurs, it follows that R must occur at β itself: cells that are stabilized at either 0 or 1 never change their value between ζ^x and Σ^x, and as a result of $\beta < \alpha$, never change between ζ^x and β, and cells that are not stabilized are 1 at all ordinals in A, and thus at β as well.

Now consider the ordinal $\alpha > \Sigma^x$, as defined before. It is impossible that α is a successor ordinal $\beta + 1$, because the snapshot at β has to be equal to the snapshot at η for some $\zeta^x \leq \eta \leq \Sigma^x$, and hence the snapshot at α must be equal to that at $\eta + 1$. So α has to be a limit ordinal. Now let δ be the limit of all ordinals ξ smaller than α such that R occurs at ξ. By the previous claim, R has to occur at δ itself, and as a result, also has to occur at $\delta + (\Sigma^x - \zeta^x)$. But this is contradictory: either $\delta + (\Sigma^x - \zeta^x)$ is smaller than α, in which case δ is not the limit we defined it as, or it is larger than or equal to α, but then there must be (because $\delta < \alpha$) some θ smaller than $(\Sigma^x - \zeta^x)$ such that $\delta + \theta = \alpha$. This however gives that the snapshot at $\delta + \theta$ is identical to the snapshot at $\zeta^x + \theta$, a contradiction. So the assumption has to be false, and $\gamma^x < \Sigma^x$ must be true. □

Finally, we note that the class of writable ordinals is downward closed, that every clockable ordinal is writable, and the classes of writable and clockable ordinals are closed under addition and multiplication:

Proposition 3.19. *The class of x-writable ordinals is closed downward, i.e. if α is x-writable, and $\beta < \alpha$, then β is x-writable. Hence, if $\alpha < \lambda^x$, then α is writable.*

Proof. See also [HaLe, No Gaps Theorem 3.7]. Consider an algorithm that writes a real x coding α. We can modify this into an algorithm writing a real coding β, by restricting x to the part isomorphic to the initial segment $\beta \subsetneq \alpha$. □

Corollary 3.20. *If α is clockable, then α is writable.*

Proof. This follows from Propositions 3.18 and 3.19. □

Proposition 3.21. *If α and β are clockable, then so are $\alpha + \beta$ and $\alpha \cdot \beta$.*

Proof. See [HaLe, Theorem 3.9]. □

Proposition 3.22. *If α and β are writable, then so are $\alpha + \beta$ and $\alpha \cdot \beta$.*

Proof. If α and β are writable, there are clockable ordinals $\eta > \alpha$ and $\theta > \beta$. We have $\eta + \theta > \alpha + \beta$ and $\eta \cdot \theta > \alpha \cdot \beta$; we also have, by Proposition 3.19, that $\eta + \theta$ and $\eta \cdot \theta$ are clockable and, hence, writable. Because the class of writable ordinals is downward closed, it follows that $\alpha + \beta$ and $\alpha \cdot \beta$ are writable. $\square$

Chapter 4

Time and space complexity

In analogy to the case of regular, finite-time Turing machines, complexity classes such as **P** and **NP** have been defined for infinite time Turing machines. In contrast to the situation for regular Turing machines, a number of important results, such as $\mathbf{P} \neq \mathbf{NP}$, have been established, sometimes with the help of descriptive set theory, in particular by reducing questions about the complexity classes to questions about analytic sets. Some main results can be found in [Sc], [DeHaSc], and [HaWe]. In this thesis, the definitions relating to nondeterministic computation and to classes such as **NP** and **NPSPACE**, as well as the major results related to those classes, will be given in chapter 7.

In this chapter we provide definitions for the deterministic classes **P** and **PSPACE** and, following that, consider to which degree these classes stay invariant between a number of different variations on the infinite time Turing machine architecture, such as the type of machine that is being used (ITTM_0 or ITTM_1), and the number of tapes and flags.

So far, the amount of literature on space complexity issues in infinite time Turing machines is rather scarce. This can be explained quite easily by the fact that, on infinite time Turing machines, almost any non-trivial computation will end up using all the ω cells of the tape. The most basic definition of space complexity—measuring the number of cells used—thus becomes meaningless. The definitions for the space complexity that will be given in this thesis will largely reflect one of the definitions given in [Lö].

4.1 Time complexity: definitions

To start with, we will give a few definitions providing a basic notation that can be used to write down the time it takes to make a certain computation for a certain Turing machine.

Definition 4.1. If T is a machine that eventually reaches the halting state q_{f}, and α is the unique ordinal such that $q_\alpha^T(x) = q_{\text{f}}$, then we say that $\mathbf{time}(x, T) = \alpha$.

Definition 4.2. For any set $A \subseteq \mathbb{R}$, we say that A is decidable by an infinite time Turing machine, if there exists an infinite time Turing machine T, such that $\phi_T(x) = 1$ for all $x \in A$, and $\phi_T(x) = 0$ for all $x \notin A$.

Definition 4.3. For any function f that assigns an ordinal to every real, and any infinite time Turing machine T, we say that T is a **time f machine** if $\mathbf{time}(x, T)$ is defined for all x and, for all $x \in \mathbb{R}$, we have $\mathbf{time}(x, T) \leq f(x)$.[1] For any ordinal ξ, we say that T is a **time ξ machine** if T is a time f machine for the constant function f with $f(x) = \xi$ for all x.

Using these definitions, we can define the notions of $\mathbf{P}_f$ machines, $\mathbf{P}_\alpha$ machines, and finally, the class $\mathbf{P}$. The following definitions originally appeared in [Sc]:

Definition 4.4. The family of all sets of reals that are decidable by a time f machine is denoted by $\mathbf{P}_f$. For any ordinal ξ, $\mathbf{P}_\xi$ denotes the family of all sets of reals that are decidable by a time η machine for some $\eta < \xi$.

Definition 4.5. The class $\mathbf{P}$ is defined as the class $\mathbf{P}_{\omega^\omega}$. Likewise, we define the class $\mathbf{P}_+$ as equal to $\mathbf{P}_f$ for the function f such that $f(x) = \omega_1^x$ for all x, and we define the class $\mathbf{P}_{++}$ as equal to $\mathbf{P}_f$ for the function f such that $f(x) = \omega_1^x + \omega + 1$ for all x. As usual, ω_1^x here stands for the least x-admissible ordinal.

Finally, we will define the class of all decidable reals:

Definition 4.6. Dec denotes the class of all sets of reals that are decidable by any infinite time Turing machine.

4.2 The weak halting problem h

Other than the main complexity classes, it will also be useful for later to have defined the weak halting problem h, as well as variants of it bounded by an ordinal specifying the time a computation is allowed to take.

Definition 4.7. We let h denote the set $\{e : \phi_e(e) \downarrow 0\}$. Furthermore, for any α, we let h_α denote the set $\{e : \phi_e(e) \downarrow 0 \wedge \mathbf{time}(e, e) \leq \alpha\}$

It should be noted that these definitions of the weak halting problem h and its relativized versions differ somewhat from the usual definitions, which are based on the mere termination of computations, rather than on termination with output 0. The main reason why this is done is that it allows for somewhat more elegant and simpler proofs of a number of theorems. For the purposes of this thesis, the definitions given here will suffice.

We have the following results about the weak halting problem and its variants bounded by an ordinal:

Proposition 4.8. $h \notin \mathbf{Dec}$

Proof. This was proven, using a similar, but different, definition of h, in [HaLe, Theorem 4.4]. Assume, towards a contradiction, that $h \in \mathbf{Dec}$. Then there must be an ITTM_0 coded by some number e, computing h, such that $\phi_e(x) \downarrow$ for all x. So, it follows that also $\phi_e(e) \downarrow$. But now it follows that $\phi_e(e) = 1$, if and only if $\phi_e(e) \downarrow 0$, if and only if $\phi_e(e) = 0$, a contradiction. So the assumption must be false, and $h \notin \mathbf{Dec}$. □

[1] For a discussion on why we used $\leq$ here (and in the later definitions), rather than $<$, see Appendix A.

Proposition 4.9. *For every ordinal α, we have $h_\alpha \notin \mathbf{P}_{\alpha+1}$.*

Proof. See also [DeHaSc, Lemma 8]. Assume, towards a contradiction, that $h_\alpha \in \mathbf{P}_{\alpha+1}$. Then there must be an ITTM_0 coded by some number e, computing h_α, such that $\phi_e(x) \downarrow$ with $\mathbf{time}(x, e) \leq \beta$ for all x, for some $\beta < \alpha + 1$. This implies that $\phi_e(x) \downarrow$ with $\mathbf{time}(x, e) \leq \alpha$ for all x, and thus also that $\phi_e(e) \downarrow$ with $\mathbf{time}(e, e) \leq \alpha$. But now it follows that $\phi_e(e) = 1$, if and only if $e \in h_\alpha$, if and only if $\phi_e(e) = 0$, a contradiction. So it must be the case that $h_\alpha \notin \mathbf{P}_{\alpha+1}$. □

The bounded variants of the halting problem, however, are still decidable in general:

Proposition 4.10. *For every writable ordinal α, we have $h_\alpha \in \mathbf{Dec}$*

Proof. See also [HaLe, Theorem 4.5]. For our h_α, the proof is easy as well: we first write α on the tape, and then, given any input i, simulate the computation of $\phi_i(i)$, using the ordinal α to count the number of steps used in the computation. Either the computation will finish before stage α, in which case we output 1 if the output of the simulation is 0, and 0 otherwise, or we will reach stage α without having finished, in which case we output 0. This algorithm outputs 1 on exactly the elements of h_α that terminate with output 0. □

4.3 Gödel's constructible hierarchy

Later on in this thesis, we will, on various occasions, need to make use of Gödel's constructible hierarchy. Its main use will be the notion of space complexity that will be defined in the next section. Here, we will restate the definitions of the hierarchy, as well as a number of basic propositions which will be of relevance later on. In doing so, we will largely follow the approach in [Ku].

First of all, we need the notion of a relativization of a formula to a certain set A:

Definition 4.11. By inductions on formulae, we define the relativization ϕ^A of all formulae ϕ to a set A as follows: (a) $(x = y)^A$ is $(x = y)$, (b) $(x \in y)^A$ is $x \in y$, (c) $(\phi \wedge \psi)^A$ is $\phi^A \wedge \psi^A$, (d) $(\neg\phi)^A$ is $\neg(\phi^A)$, and (e) $(\exists x\phi)^A$ is $\exists x(x \in A \wedge \phi^A)$.

Now we can define the *definable power set* operation $\mathcal{D}$ in the following way:

Definition 4.12. $\mathcal{D}(A) := \{B : \text{There is a formula } \phi(y_1, \ldots, y_n, x) \text{ using only constants from } A, \text{ and with all free variables shown, and there are elements } v_1, \ldots, v_n \in A \text{ such that } B = \{x \in A : \phi^A(v_1, \ldots, v_n, x)\}\}$

The idea here is that $\mathcal{D}(A)$ contains all subsets of A which are definable from a finite number of elements of A by a formula relativized to A.

Definition 4.13. Now we can define the constructible hierarchy relative to a set x in the following way using transfinite recursion:

1. $\mathbf{L}_0[x] = \mathrm{TC}(x)$ (where $\mathrm{TC}(x)$ is the transitive closure of x),
2. $\mathbf{L}_{\alpha+1}[x] = \mathcal{D}(\mathbf{L}_\alpha[x])$, and
3. $\mathbf{L}_\alpha[x] = \bigcup_{\xi<\alpha} \mathbf{L}_\xi[x]$ when α is a limit ordinal.

Lemma 4.14. *For any ordinal α and any set w, $\mathbf{L}_\alpha[w]$ is transitive.*

Proof. We prove this by ordinal induction. For the base case where $\alpha = 0$, observe that $\mathbf{L}_0[w]$ is defined as the transitive closure of w, and is hence transitive.

In the case where $\alpha = \beta + 1$, consider any element $x \in \mathbf{L}_\alpha[w]$, and an element $y \in x$. We have to show that $y \in \mathbf{L}_\alpha[w]$. By definition, because $\mathbf{L}_\alpha[w] = \mathcal{D}(\mathbf{L}_\beta[w])$, it follows that $x \subseteq \mathbf{L}_\beta[w]$, and hence that $y \in \mathbf{L}_\beta[w]$. By the inductive hypothesis, $\mathbf{L}_\beta[w]$ itself is transitive, and thus, for all $z \in y$, we have that $z \in \mathbf{L}_\beta[w]$. It now follows that we can define y in $\mathbf{L}_0[\alpha]$ as $\{x \in \mathbf{L}_\beta[w] : x \in y\}$. Hence, $\mathbf{L}_0[\alpha]$ is transitive.

In the case where α is a limit ordinal, consider any element $x \in \mathbf{L}_\alpha[w]$. By definition, there has to be a $\xi < \alpha$, such that $x \in \mathbf{L}_\xi[w]$. For any $y \in x$, we have, by the inductive hypothesis, that $y \in \mathbf{L}_\xi[w]$, and hence that $y \in \mathbf{L}_\alpha[w]$, completing the proof. □

Lemma 4.15. *For any two ordinals α, β, such that $\alpha < \beta$, and any set w, we have that if $x \in \mathbf{L}_\alpha[w]$, then also $x \in \mathbf{L}_\beta[w]$.*

Proof. When β is a limit ordinal, this is a direct consequence of the definition. When β is a successor ordinal, we either have that there is a natural number n such that $\beta = \alpha + n$, or we have that there is a limit ordinal γ and a natural number n such that $\alpha < \gamma$ and $\beta = \gamma + n$.

Now, observe that if $x \in \mathbf{L}_\alpha[w]$ for any ordinal α, we have, by the transitivity of $\mathbf{L}_\alpha[w]$, that $x \subseteq \mathbf{L}_\alpha[w]$, and hence we have that x is definable from $\mathbf{L}_\alpha[w]$: $x = \{y \in \mathbf{L}_\alpha[w] : y \in x\}$. So we have that $x \in \mathbf{L}_{\alpha+1}[w]$.

From this it follows inductively that if $x \in \mathbf{L}_\alpha[w]$, then, for any natural number n, $x \in \mathbf{L}_{\alpha+n}[w]$, completing the proof in both of the named cases. □

Lemma 4.16. *For any ordinal α and any set w, $\alpha \in \mathbf{L}_{\alpha+1}[w]$.*

Proof. The proof goes by transfinite recursion. We distinguish the following three cases:

1. α is 0: We have that $0 = \emptyset = \{x \in w : x \neq x\}$ which is in $\mathcal{D}(w)$ by definition, and hence in $\mathbf{L}_1[w]$.

2. α is $\beta + 1$: Assume that $\beta \in \mathbf{L}_{\beta+1}[w]$. Then we have (by the regular definition of ordinals) that $\alpha = \beta \cup \{\beta\}$, and we can now see that $\alpha = \{x \in \mathbf{L}_\alpha[w] : x = \beta \vee x \in \beta\}$, and hence $\alpha \in \mathbf{L}_{\alpha+1}[w]$.

3. α is a limit ordinal: clearly we have for any $\beta < \alpha$ that $\beta \in \mathbf{L}_{\beta+1}[w]$, and hence that $\beta \in \mathbf{L}_\alpha[w]$. We can now distinguish two situations: either we already have $\alpha \in \mathbf{L}_\alpha[w]$, in which case we surely have $\alpha \in \mathbf{L}_{\alpha+1}[w]$, or we have that $\alpha = \{x \in \mathbf{L}_\alpha[w] : x \text{ is an ordinal}\}$, in which case we have a definition of α from $\mathbf{L}_\alpha[w]$, and hence also $\alpha \in \mathbf{L}_{\alpha+1}[w]$. □

We now turn to the observation, that if a certain subset of ω occurs at a certain level $\alpha > \omega + 1$ in the constructible hierarchy, any other subset of ω that differs from this set by only finitely many elements occurs at the same level α.

Lemma 4.17. *If $\alpha > \omega$, and $A \subset \omega$, and $A \in \mathbf{L}_\alpha[w]$ for a certain set w, and for a certain set B, $|(A \backslash B) \cup (B \backslash A)| < \omega$, then $B \in \mathbf{L}_\alpha[w]$.*

Proof. We may assume that α is a successor ordinal. To see why, consider that, if α is a limit ordinal, and $A \in \mathbf{L}_\alpha[w]$, there has to be a successor ordinal $\beta < \alpha$, such that $A \in \mathbf{L}_\beta[w]$, and if we have $B \in \mathbf{L}_\beta[w]$, it directly follows that $B \in \mathbf{L}_\alpha[w]$.

Now let $B \backslash A = \{b_1, \ldots, b_m\}$, and let $A \backslash B = \{a_1, \ldots a_n\}$, and let us say that $\alpha = \gamma + 1$. We have, by definition, that $A \in \mathcal{D}(\mathbf{L}_\gamma[w])$, and hence there is a formula ϕ and elements $v_1, \ldots, v_k$ such that $A = \{x \in \mathbf{L}_\gamma[w] : \phi^A(v_1, \ldots, v_k, x)\}$. But now we can also define B from $\mathbf{L}_\gamma[w]$ as follows: $B = \{x \in \mathbf{L}_\gamma[w] : (\phi^A(v_1, \ldots, v_k, x) \vee (x = b_1 \vee \ldots \vee x = b_m)) \wedge \neg(x = a_1 \vee \ldots \vee x = a_n)\}$. From this, it follows that also $B \in \mathbf{L}_\alpha[w]$. □

4.4 Space complexity: definitions

With regards to time complexity, the definitions of the classes were relatively natural: like in the case of finite Turing machines, we could simply look at the amount of time in which sets could be decided. In the case of space complexity, things turn out to be less simple. The most obvious way to define space complexity—simply by counting the number of cells used in the computation—appears to be quite pointless here, as almost all computations except for a few trivial ones, would take up all of the ω cells of the tapes.

A reasonable alternative to simply counting the number of cells used, and the alternative chosen in [Lö], is to look at the complexity of the reals that are written on the tape. This could be done in various ways—one of them being Gödel's constructible **L**-hierarchy that we defined in section 4.3: to compute the space complexity of a computation starting from a certain input x we could, for example, look at the levels of the hierarchy relativized to x, look at which level all the snapshots of the tapes appear, and define the space complexity of a computation as the ordinal corresponding to the level of the 'most complicated' real that occurs during the computation. More precisely:

Definition 4.18. We define:

1. $\ell^T_\alpha(x) := \min\{\eta : \sup\{c^T_{i,\alpha} : i \in \{1, 2, 3\}\} \in \mathbf{L}_\eta[x]\}$, and
2. $\mathbf{space}(x, T) := \sup\{\ell^T_\xi : \xi \leq \mathbf{time}(x, T)\}$

With this we can, analogously to the case of time complexity, define space f machines for any f, and, from there, the space classes $\mathbf{PSPACE}_f$, $\mathbf{PSPACE}_\alpha$, etc.:

Definition 4.19. For any infinite time Turing machine T, we say that T is a **space f machine** if, for all $x \in \mathbb{R}$, we have $\mathbf{space}(x, T) \leq f(x)$. For any ordinal ξ, we say that T is a **space ξ machine** if T is a space f machine for the constant function f such that $f(x) = \xi$ for all x.

Definition 4.20. For any function f, we let $\mathbf{PSPACE}_f$ denote the class of all sets of reals that are decidable by a space f machine. For any ordinal ξ, we let $\mathbf{PSPACE}_\xi$ denote the class of all sets of reals that are decidable by a space η machine for some $\eta < \xi$.

Definition 4.21. The class **PSPACE** is defined as the class $\mathbf{PSPACE}_{\omega^\omega}$. Likewise, we define the class $\mathbf{PSPACE}_+$ as equal to $\mathbf{PSPACE}_f$ for the function

f such that $f(x) = \omega_1^x$ for all x, and the class $\mathbf{PSPACE}_{++}$ is defined as $\mathbf{PSPACE}_f$ for the function f such that $f(x) = \omega_1^x + \omega + 1$ for all x.

4.5 Invariance under different ITTM-models

4.5.1 Invariance of time complexity

In chapter 2, we have introduced different variants of the infinite time Turing machine architecture: the ITTM_0 type as well as the ITTM_1 type, and beyond that, we have introduced variants with a varying number of tapes and flags. This leads us to wonder whether the space and time complexity classes in this chapter are invariant between these types of machines. As it turns out, this equivalence can be shown for most limit ordinals between most different types of machines. To show this, we first give two propositions that establish the invariance of time complexity, at limit ordinals, between machines with three tapes, and machines with extra tapes and/or flags.

In the following two propositions, simulations similar, but not identical, to those given in chapter 2 will be provided. The main difference with the simulations in chapter 2 is that, this time, we set up the simulation such that no contraction operation is necessary. We do this by ensuring that the only tape being simulated on the second tape of the simulation, is the original second tape, and ensuring that the first cell of tape 2 of the simulation corresponds to the first cell of the original tape 2. Combined with the fact that we are considering machines deciding sets, i.e. machines with $\emptyset$ and $\{0\}$ (a.k.a. 0 and 1) as only output values, this eliminates the need for a contraction operation. Other than this, the stretching part of the simulation provided differs from the ones in chapter 2, in the sense that here we put a stretched copy of the first tape on the third tape, with the help of the second tape: this can be done in time ω, rather than the ω^2 needed for the stretching operation of Lemma 2.20. Finally, the actual simulation here simulates n tapes on 1 tape, instead of $2n$ tapes on n tapes.

Proposition 4.22. *If α is a limit ordinal larger than or equal to ω^2, and we have some $n \geq 3$ and some $m \geq 0$, then any set A can be decided by an* ITTM_0 *with 3 tapes in time β for some $\beta < \alpha$, if and only if it can be decided by an* ITTM_0 *with n tapes and m flags in time η for some $\eta < \alpha$.*

Proof. We can simulate a machine with n tapes by a machine with 3 tapes in the following way: we first put a stretched copy of the input tape on the third tape, leaving $n-2$ empty cells between each cell of the input tape, while filling the first tape with zeros. This can be performed in ω steps if we make use of the second tape as auxiliary tape while performing this operation. When we find ourself in the limit state, reading a 0 on the first tape, we know that the preparatory work has been finished, and the actual simulation can begin. We now write a 1 on the first tape, never to change afterwards, and start simulating the original computation, with all the n tapes except for the second one being simulated on the third tape, and the second tape of the original tape being simulated on the positions $0, 0+n, 0+2n, \ldots, 0+kn, \ldots$ of the second tape. This way, we will end up having a 1 on the first position of the output tape if and only if we have a 1 on the first position of the simulated output tape.

Moreover, each step in the original computation is now simulated by a fixed finite number of steps. In this simulation, only one new limit is introduced, at the very beginning, and this extra limit becomes irrelevant from the point ω^2 onward.

Furthermore, we can simulate flags with extra tapes, again without introducing new limits. So it is clear that a machine with $n \geq 3$ tapes and $m \geq 0$ flags can be simulated by a machine with three tapes, without introducing any new limits. However, we still need to make sure that, also in the case of flags, we actually stay uniformly below α: after all, the task of reading/writing the simulated flag cell involves moving the head to the first cell of the tape and back again, and the number of steps it takes to do this is not fixed, but rather depends on the position of the head. If α is a limit ordinal larger than ω^2, consider the following: say, the simulated computation takes at most $\theta + n$ steps, where $\theta \geq \omega^2$ is a limit ordinal and n is a natural number. The first θ steps are simulated in exactly θ steps, as no new limits are introduced beyond ω^2, and at stage θ, the position of the head is 0. As a result, during the last n steps, the position of the head will always be smaller than or equal to n. We now let k denote the maximum number of steps it can take to move from a head position $\leq n$ to 0 to read/write the flag, and back again. We now are sure that the simulating computation takes at most $\theta + k \cdot n$ steps, so we are indeed uniformly below α. In the case where α itself is ω^2, we can use a similar kind of reasoning: the difference here is that the simulating machine takes an additional ω steps, but it will still stay uniformly below ω^2.

For the other direction, we can simply ignore the extra tapes and flags, and thus carry out the computation in exactly the same time as the original computation. □

Proposition 4.23. *If α is a limit ordinal larger than or equal to ω^2, and we have some $n \geq 3$ and some $m \geq 0$, then any set A can be decided by an* ITTM_1 *with 3 tapes in time β for some $\beta < \alpha$, if and only if it can be decided by an* ITTM_1 *with n tapes and m scratch cells in time η for some $\eta < \alpha$.*

Proof. Here we can use the same tactic as in the Proposition 4.22; we only need to take into account the fact that, after the first ω steps, we now will not find ourselves in a limit state, but rather in the highest-numbered state that has been visited cofinally often. We can solve this by again using a 'signaling' state that will always read a 1 somewhere (for example on the first tape) in successor states, but that will read a 0 in a limit state. Once we reach this signaling state, reading a 0, we go on to the actual simulated computation, and the further details of the proof are be identical to those of Proposition 4.22. □

We now turn to the invariance of time complexity, again at limit ordinals, between ITTM_0s and ITTM_1s:

Proposition 4.24. *If α is a limit ordinal larger than or equal to ω^2, and the set A can be decided by an* ITTM_0 *machine (with three tapes) in time β for some $\beta < \alpha$, then it can be decided by an* ITTM_1 *machine (with three tapes) in time η for some $\eta < \alpha$.*

Proof. We start by noting that, in the simulation described in Proposition 2.30, no new limits are introduced and, in fact, there is a one-to-one equivalence of the limits of the simulating machine and the simulated machine. So we know

that any ITTM_0 machine can be simulated by an ITTM_1 machine with extra flags while staying uniformly below the same limits; from Proposition 4.22, it now follows, that this machine can also be simulated by an ITTM_1 machine with 3 tapes and no flags. □

Proposition 4.25. *If α is a limit ordinal larger than or equal to ω^2, and the set A can be decided by an* ITTM_1 *machine in time β for some $\beta < \alpha$, then it can be decided by an* ITTM_0 *machine in time η for some $\eta < \alpha$.*

Proof. Looking at Proposition 2.31, we can draw much of the same conclusions as in Proposition 4.24, with the difference that in this case, we actually only need one single flag. Again, there are no new limits introduced during the simulation itself, and we only need to introduce a single new limit at the very beginning, which already becomes irrelevant at stage ω^2. □

So, as it turns out, there is an equivalence, at the level of all limit ordinals from ω^2 onward, between a wide range of differently defined machines: ITTM_0 and ITTM_1-machines with three or more tapes and an arbitrary finite number of flags are all effectively equivalent.

4.5.2 Invariance of space complexity

As we have seen, we have a reasonable degree of invariance of time complexity under the different models and varying numbers of tapes. We now turn to the question whether the same thing holds for space complexity. It turns out that, again, we have such an invariance, this time not only for limit ordinals but also for successor ordinals. For this proof, we first need two auxiliary lemmata:

Lemma 4.26. *For any number $n \in \mathbb{N}$, any $k < 0$ and any set w, the set corresponding to the function $f : \mathbb{N} \to \mathbb{N}$ such that $f(m) = nm + k$ for all m is in $\mathbf{L}_{\omega+1}[w]$.*

Proof. It is easily checked that, for all sets A, B, and C, the properties $A = (B, C)$, $A = B \times C$, $A = B \uplus C$ (i.e. $A = (\{\emptyset\} \times B) \cup (\{\{\emptyset\}\} \times C)$), '$A$ is an ordinal', and, for finite sets, 'A is equinumerous with B' are all describable by first order formulae relativized to $\mathbf{L}_\omega[w]$. So, the notions $\alpha + \beta$ and $\alpha \cdot \beta$ are describable as, 'The ordinal equinumerous with $\alpha \uplus \beta$' and 'The ordinal equinumerous with $\alpha \times \beta$', respectively. As a result, we can describe f with the description: 'The family of sets of the type (x, y), such that x and y are (finite) ordinals and $y = x \cdot n + k$'. □

Lemma 4.27. *For any family of sets $A_1, \ldots, A_n$, and any ordinal $\alpha > \omega$, the following holds: if $A_i \in \mathbf{L}_\alpha[w]$ for every i, then the set B such that $(n \cdot k + (i - 1)) \in B \leftrightarrow k \in A_i$ is also in $\mathbf{L}_\alpha[w]$.*

Proof. We can describe B as the set of (finite) ordinals x such that there are ordinals k and i such that $k \in A_i$ and $x = n \cdot k + (i - 1)$. If α is a successor ordinal $\beta + 1$, it can easily be checked that these notions are describable by a formula relativized to $\mathbf{L}_\beta[w]$. If α is a limit ordinal, there must be some successor ordinal $\beta < \alpha$ such that $A_i \in \mathbf{L}_\beta[w]$ for every i, and it follows that B is in $\mathbf{L}_\beta[w]$ and, hence, also must be in $\mathbf{L}_\alpha[w]$. □

With these lemmata, it is easy to establish the invariance of the defined space complexity classes between ITTM_0 and ITTM_1, and between the number n of tapes used for any $n \geq 3$.

Proposition 4.28. *If T is an ITTM_0 with $n \geq 3$ tapes and $m \geq 0$ flags, deciding some set A, there is an ITTM_0 S with 3 tapes, also deciding A, such that $\mathbf{space}(x, T) = \mathbf{space}(x, S)$ for every real x.*

Proof. We consider the simulation of T on a three tape machine, as described in Proposition 4.22, that first puts a stretched copy of the first tape on the third tape, then simulates the computation of T using the second tape to simulate the original second tape, and the third tape to simulate all other tapes.

Using Lemmata 4.26 and 4.27, it is easy to see that the space complexity of the simulation is identical to the space complexity of the original computation under the simulation described in Proposition 4.22. □

A consequence from this theorem is the fact, that flags never increase the space complexity of a computation, at least as long as this complexity is already as least as large as ω, simply because flags can be considered as extra tapes with an extremely low (finite) complexity, which are always in $\mathbf{L}_\omega[x]$ for every x.

Proposition 4.29. *If T is an ITTM_0 with 3 tapes, deciding some set A, there is an ITTM_1 S with 3 tapes, also deciding A, such that $\mathbf{space}(x, T) = \mathbf{space}(x, S)$ for every real x.*

Proof. If we perform the simulation described in 2.31, we can note that this is essentially a computation that only introduces a finite number of new flags and, other than that, will have exactly the same content on the three main tapes as the simulated computation. Because, as we have seen, flags will never introduce any extra space complexity, the space complexity of the simulating computation will be identical to that of the simulated computation. □

Proposition 4.30. *If T is an ITTM_1 with $n \geq 3$ tapes and $m \geq 0$ flags, deciding some set A, there is an ITTM_1 S with 3 tapes, also deciding A, such that $\mathbf{space}(x, T) = \mathbf{space}(x, S)$ for every real x.*

Proof. This goes exactly analogous to 4.28. □

Here again, it should be noted that, as a result, flags never increase the space complexity of a computation, as long as this complexity is already larger than or equal to ω.

Proposition 4.31. *If T is an ITTM_1 with 3 tapes, deciding some set A, there is an ITTM_0 S with 3 tapes, also deciding A, such that $\mathbf{space}(x, T) = \mathbf{space}(x, S)$ for every real x.*

Proof. Here we can again make the same observation as in 4.29: the simulation as described in 2.30 only introduces a finite number of flags, and, other than that, will have the exact same tape contents on the three main tapes. □

From the above we can conclude that, at least in the case of ordinals larger than ω, the exact type of machine used makes no difference whatsoever with regard to which space complexity classes a certain set A does and does not belong to. To show that a certain set is, or is not, in a certain space complexity class, we may therefore use any machine we like, with an arbitrary number of extra tapes and an arbitrary number of flags/scratch cells.

Chapter 5

The question $\mathbf{P} \stackrel{?}{=} \mathbf{PSPACE}$

5.1 $\mathbf{P} \subseteq \mathbf{PSPACE}$ for infinite time Turing machines

First of all, we want to be sure that **P** is always a subset of **PSPACE**. Due to the somewhat unusual definition of **PSPACE**, however, it is not directly evident that it is indeed true that the space complexity of a computation will always be equal to or smaller than the time complexity. In this section, it will be shown that this indeed is the case. This will be done by showing that we can represent any ITTM-computation of length α, starting from a set w, within the level $\mathbf{L}_\alpha[w]$ of the constructible hierarchy. We will first define what we mean with a 'representation' of an ITTM computation:

Definition 5.1. We use the notation $C^T_\alpha(w)$ for a representation of an ITTM computation of a machine T from w in α steps, of the form

$$C^T_\alpha(w) = \{(\beta, q^T_\beta(w), h^T_\beta(w), c^T_{1,\beta}(w), c^T_{2,\beta}(2), c^T_{3,\beta}(w)) : \beta \leq \alpha\}$$

where β is an ordinal representing the stage, and $q^T_\beta(w)$, $h^T_\beta(w)$, $c^T_{i,\beta}(w)$ are as they were defined earlier.

Lemma 5.2. *When we have a suitable representation of Turing machines as finite sets, the notion 'a one step ITTM computation by a machine T from the state $t = (q, h, c_1, c_2, c_3)$ results in a state $t' = (q', h', c'_1, c'_2, c'_3)$ is representable by a formula of first order logic.*

Proof. Let us take the following representation of Turing machines: a Turing machine T is a set of tuples of the form $(q, d_1, d_2, d_3, m, d'_1, d'_2, d'_3, q')$, where q is the current state, q' is the new state, both represented as a natural number, d_1, d_2, d_3 are the symbols read on the tape (either 0 or 1), m is the direction in which the Turing machine will move (either 0, left, or 1, right), and d'_1, d'_2, d'_3 are the new contents of the tapes at position h.

With this formulation, we can represent the above notion as: 'There are $q, d_1, d_2, d_3, m, d'_1, d'_2, d'_3, q'$ such that $(q, d_1, d_2, d_3, m, d'_1, d'_2, d'_3, q') \in T$, there are h, c_1, c_2, c_3 such that $t = (q, h, c_1, c_2, c_3)$, and there are h', c'_1, c'_2, c'_3 such that $t' = (q', h', c'_1, c'_2, c'_3)$, for which the following further properties hold: if m equals

0, $h' = h - 1$, or, if $h = 0$, then $h' = 0$, and if m equals 1, $h' = h + 1$; for each d_i, $h \in c_i$ if and only if $d_i = 1$, and for each d'_i, $h \in c'_i$ if and only if $d'_i = 1$.' This can be written as a (rather long, which is why it is avoided here) first-order formula without too much further thinking work. □

Lemma 5.3. *The notion '$X = C^T_\alpha(w)$: X represents an ITTM computation from w in α steps' is representable by a formula of first order logic.*

Proof. Note that X represents an ITTM computation from w in α steps, if and only if:

1. Every element of X is of the form $(\beta, q, h, c_1, c_2, c_3)$ where β is an ordinal smaller than or equal to α.
2. If we have $(\beta, q_1, h_1, c_{1,1}, c_{2,1}, c_{3,1}) \in X$ and $(\beta, q_2, h_2, c_{1,2}, c_{2,2}, c_{3,2}) \in X$, then $q_1 = q_2$, $h_1 = h_2$, $c_{1,1} = c_{1,2}$, $c_{2,1} = c_{2,2}$ and $c_{3,1} = c_{3,2}$.
3. We have $(0, q_\mathrm{s}, h_0, c_{1,0}, c_{2,0}, c_{3,0})$ where q_s is the initial state of the Turing machine, $h_0 = 0$, $c_{1,0} = w$, $c_{2,0} = c_{3,0} = \emptyset$.
4. If β is a successor ordinal $\gamma + 1$, and $\beta < \alpha$, then there are elements $(\gamma, q_1, h_1, c_{1,1}, c_{2,1}, c_{3,1})$ and $(\beta, q_2, h_2, c_{1,2}, c_{2,2}, c_{3,2})$ such that a one step Turing computation from the snapshot $(q_1, h_1, c_{1,1}, c_{2,1}, c_{3,1})$ results in $(q_2, h_2, c_{1,2}, c_{2,2}, c_{3,2})$.
5. If β is a limit ordinal, then there is an element $\beta, q, h, c_1, c_2, c_3$ such that q is the limit state of the Turing machine, $h = 0$, and for each of c_i, we have that $x \in c_i$ if and only if for every ordinal $\gamma < \beta$ there is an ordinal δ greater than or equal to γ and smaller than β such that if $(\delta, q', h', c'_1, c'_2, c'_3) \in X$, then $x \in c'_i$.

It should be clear that all of these notions are representable by a formula of first order logic in the language of set theory. □

Now we will turn to the main theorem. The final aim will be to show, that all computations starting from an input w of length α can be carried out while only having tape contents slightly more complicated than in $\mathbf{L}_\alpha[w]$. The addition of 'slightly' here only signifies that one needs a small finite fixed extra number of steps—the construction made here assumes gives a crude upper bound of 12, although the precise number might be lower than that. First, we need an additional auxiliary lemma:

Lemma 5.4. *If $\beta, q, h, c_1, c_2, c_3$ are all in $\mathbf{L}_\alpha[w]$, then $(\beta, q, h, c_1, c_2, c_3)$ is in $\mathbf{L}_{\alpha+10}[w]$.*

Proof. We assume that (c_2, c_3) is defined as $\{\{c_2\}, \{c_2, c_3\}\}$. If c_2 and c_3 are in $\mathbf{L}_\alpha$, then $\{c_2\}$ and $\{c_2, c_3\}$ are definable subsets of $\mathbf{L}_\alpha$, and are thus in $\mathbf{L}_{\alpha+1}$. Consequently, $\{\{c_2\}, \{c_2, c_3\}\}$ is a definable subset of $\mathbf{L}_{\alpha+1}$, and is hence in $\mathbf{L}_{\alpha+2}$.

Assuming that $(\beta, q, h, c_1, c_2, c_3)$ is defined as $(\beta, (q, (h, (c_1, (c_2, c_3)))))$, it easily follows by repeating the above procedure that $(\beta, q, h, c_1, c_2, c_3)$ is in $\mathbf{L}_{\alpha+10}$. □

Now we will turn to the main theorem, which will be proved using a kind of simultaneous induction:

Theorem 5.5. *The following propositions are true:*

1. *For any infinite ordinal α and any input set $w \subseteq \omega$, we have that $c^T_{1,\alpha}(w)$, $c^T_{2,\alpha}(w)$, $c^T_{3,\alpha}(w)$ are in $\mathbf{L}_{\alpha+1}[w]$. If α is a finite ordinal, however, we can only be sure that the sets are in $\mathbf{L}_{\alpha+2}[w]$.*

2. *If α is a successor ordinal above ω and is equal to $\beta + n$, where n is a natural number, then $c^T_{1,\alpha}(w)$, $c^T_{2,\alpha}(w)$, $c^T_{3,\alpha}(w)$ are in $\mathbf{L}_{\beta+1}[w]$. This is a strengthening of the above property.*

3. *For any ordinal $\alpha \geq \omega$ and any input set $w \subseteq \omega$, we have $(\alpha, q^T_\alpha(w), h^T_\alpha(w), c^T_{1,\alpha}(w), c^T_{2,\alpha}(w), c^T_{3,\alpha}(w)) \in \mathbf{L}_{\alpha+11}[w]$.*

4. *For any ordinal $\alpha \geq \omega$ and any input set $w \subseteq \omega$, we have $C^T_\alpha(w) \in \mathbf{L}_{\alpha+12}[w]$.*

Proof. The tactic here will be to first show that, for any α, if property (1) holds for all ordinals smaller than or equal to α, and property (4) holds for all ordinals strictly smaller than α, then properties (2), (3), and (4) hold for α. Then, we will show that if properties (1), (2), (3), and (4) hold for all ordinals strictly smaller than α, then property (1) also holds for α.

- $1 \rightarrow 2$: We use here the fact that, assuming $\alpha > \omega$, and $\alpha = \beta + n$, for each i in $\{1, 2, 3\}$, $c^T_{i,\beta}(w)$ and $c^T_{i,\alpha}(w)$ can only differ by finitely many elements. Using 4.17, it follows from $c^T_{i,\beta}(w) \in \mathbf{L}_{\beta+1}[w]$ that $c^T_{i,\alpha}(w) \in \mathbf{L}_{\beta+1}[w]$.

- $1 \rightarrow 3$: By Lemma 4.16, we have that $\alpha \in \mathbf{L}_{\alpha+1}[w]$ for any w. Because $q^T_\alpha(w)$ and $h^T_\alpha(w)$ are finite sets, they are also in $\mathbf{L}_{\alpha+1}[w]$ by the assumption that α is infinite, and the desired result follows from the assumption of (1) and an application of Lemma 5.4.

- 1 $(+3) \rightarrow 4$ for successor ordinals: If α is a successor ordinal $\beta + 1$, then we have that x is in $C^T_\alpha(w)$ if and only if $x \in C^T_\beta(w)$ or if x is equal to $(\alpha, q^T_\alpha(w), h^T_\alpha(w), c^T_{1,\alpha}(w), c^T_{2,\alpha}(w), c^T_{3,\alpha}(w))$. Because by the inductive hypothesis $C^T_\beta(w)$ and hence, by transitivity, every element of it, is in $\mathbf{L}_{\beta+12}[w]$, or $\mathbf{L}_{\alpha+11}[w]$, and because by (3)—which was already proven from (1)—$(\alpha, q^T_\alpha(w), h^t_\alpha(w), c^T_{1,\alpha}(w), c^T_{2,\alpha}(w), c^T_{3,\alpha}(w))$ is in $\mathbf{L}_{\alpha+11}$, it follows that $C^T_\alpha(w)$ is a definable subset of $\mathbf{L}_{\alpha+11}[w]$, and hence an element of $\mathbf{L}_{\alpha+12}[w]$.

- 1 $(+3) \rightarrow 4$ for limit ordinals: If α is a limit ordinal, note that, for any $\beta < \alpha$, we have by the inductive hypothesis that $C^T_\beta(w) \in \mathbf{L}_{\beta+12}[w]$, and hence also that $C^T_\beta(w) \in \mathbf{L}_\alpha[w]$. Now consider the following set $C^T_{<\alpha}(w)$:

  $$\{X \in L_\alpha[w] : \exists\beta(\beta \in \alpha \wedge X \in C^T_\beta(w))\}$$

  Note that $\beta \in \alpha$ basically just says, 'β is an ordinal smaller than α'. This set is clearly a set definable from $L_\alpha[w]$, and it is the union of all computations from w in less than α steps. To obtain $C^T_\alpha(w)$, we only need to add $(\alpha, q^T_\alpha(w), h^t_\alpha(w), c^T_{1,\alpha}(w), c^T_{2,\alpha}(w), c^T_{3,\alpha}(w))$ to this set, which can be done easily like in the above case of successor ordinals.

- Property (1) for finite ordinals: note that, at stage n, because the head starts at 0 and can only move forward one step at a time, the head can only be at a location between 0 and n. Because of this, the only positions of $c^T_{i,n}$ that can have changed by stage n are the positions in the range $[0, n-1]$. Because we have $c^T_{0,0} = w$ and $c^T_{1,0} = c^T_{2,0} = \emptyset$, we also have $c^T_{i,0} \in L_1[w]$.

  Now, we can define $c^T_{i,n}$ as $(c^T_{i,0} \backslash \{j_1, \ldots, j_l\}) \cup \{k_1, \ldots, k_m\}$, which is a definable subset of $L_n[w]$ as a result of the fact that all the j and k are not larger than $n-1$ and are thus elements of $L_n[w]$.

- Inductive case for successor ordinals larger than ω: If α is a successor ordinal $\beta + 1$, for each $i \in \{1, 2, 3\}$ $c^T_{i,\alpha}(w)$ and $c^T_{i,\beta}(w)$ can only differ by one element. Taking the largest limit ordinal γ below α, such that $\alpha = \gamma + n$, an appeal to Lemma 4.17 (and possibly to case (2) for β, if β itself is a successor ordinal) suffices to show that for each $i \in \{1, 2, 3\}$, $c^T_{i,\alpha}$ is in $\mathbf{L}_{\gamma+1}[w]$ and hence also in $\mathbf{L}_{\alpha+1}[w]$.

- Inductive case for limit ordinals: if α is a limit ordinal, we have, by inductive assumption, that for all $\beta < \alpha$, $C^T_\beta(w) \in \mathbf{L}_{\beta+12}[w]$, and hence, also that $C^T_\beta(w) \in \mathbf{L}_\alpha[w]$.

  To start, let us find a way of identifying ordinals smaller than α in $\mathbf{L}_\alpha[w]$. If $\alpha \notin \mathbf{L}_\alpha[w]$, the set of ordinals in $L_\alpha[w]$ is exactly the set of ordinals smaller than α: in this case we can identify any ordinal smaller than α in $\mathbf{L}_\alpha[w]$ with the property $\mathrm{Ord}(x)$. If somehow we do have $\alpha \in \mathbf{L}_\alpha[w]$, we can identify any ordinal smaller than α in $L_\alpha[w]$ with the property $x \in \alpha$. Depending on which case we are in, let us take the appropriate property.

  Now note that the definition of cells at limit values implies that $x \in c^T_{i,\alpha}(w)$ if and only if for every ordinal $\beta < \alpha$, there is an ordinal $\gamma < \alpha$ with $\gamma \geq \beta$, such that $x \in c^T_{i\gamma}(w)$. We can now define $c^T_{i,\alpha}(w)$ in the following way:

  $c^T_{i,\alpha}(w)$ is the set of all $n \in \omega$, such that for every ordinal $\beta < \alpha$ there is an ordinal $\gamma < \alpha$ with $\gamma \geq \beta$, such that n is an element of the $i+3$th argument of $C^T_\gamma(w)$ (i.e. $n \in c^T_{i,\gamma}(w)$).

Although it should be noted that we have proved property (3) and (4) only for infinite ordinals, this does not pose a problem, because for any finite n, we have that $C^T_n(w)$ will occur at at least some finite stage of Gödel's Constructible hierarchy. As a result, we will have $C^T_n(w) \in \mathbf{L}_\omega[w]$ for any finite n, and the inductive steps will still work for the ω-case. □

This directly gives us the following result about **P** and **PSPACE**:

Corollary 5.6. *For any function f, we have $\mathbf{P}_f \subseteq \mathbf{PSPACE}_{f+2}$, where $f+2$ is the function such that $(f+2)(x) = f(x) + 2$ for all x.*

Proof. Let T be a machine deciding A in time f. That means that for all inputs x, the computation has length $\mathbf{time}(x, T) \leq f(x)$. Fix some $\xi \leq \mathbf{time}(x, T) \leq f(x)$. By Theorem 5.5, we have $c^T_{i,\xi}(x) \in \mathbf{L}_{\xi+2}[x]$ for $i \in \{1, 2, 3\}$, and hence $c^T_{i,\xi}(x) \in \mathbf{L}_{f(x)+2}[x]$. □

Indeed, we almost have $\mathbf{P}_f \subseteq \mathbf{PSPACE}_f$: the obstacle preventing this result consists of the finite computation stages n, at which we only have shown $c^T_{i,n}(w) \in \mathbf{L}_{n+2}[w]$, rather than the desired $c^T_{i,n}(w) \in \mathbf{L}_{n+1}[w]$.

Proposition 5.7. *For any infinite time Turing machine T and any real x, if $\mathbf{time}(x,T) > \omega$, we have $\mathbf{space}(x,T) \leq \mathbf{time}(x,T)$. Also, if $\mathbf{time}(x,T) \leq \omega$, we have $\mathbf{space}(x,T) \leq \omega$. Hence, for any function f such that $f(x) > \omega$ for all x, every time f machine is a space f machine, and consequently we have $\mathbf{P}_f \subseteq \mathbf{PSPACE}_f$.*

Proof. Assume that $\alpha > \omega$ and $\alpha = \mathbf{time}(x,T)$. For any $\xi < \alpha$ with $\xi \geq \omega$, we have, by Theorem 5.5, $c^T_{i,\xi}(x) \in \mathbf{L}_{\xi+1}[x]$ (here we have $+1$ instead of $+2$ because $\xi \geq \omega$) for $i \in \{1,2,3\}$, and hence $c^T_{i,\xi}(x) \in \mathbf{L}_\alpha[x]$. For finite ξ, we have $c^T_{i,\xi}(x) \in \mathbf{L}_\omega[x]$ for $i \in \{1,2,3\}$, and again $c^T_{i,\xi}(x) \in \mathbf{L}_\alpha[x]$. Finally, we notice that α must be equal to $\beta+1$ for some β, as infinite time Turing machines cannot halt at limit ordinal stages. It is also immediate from the definition of infinite time Turing machines that $c^T_{i,\beta}(x)$ and $c^T_{i,\beta+1}(x)$ can only differ by one element at most. Because of this, it follows from $c^T_{i,\beta}(x) \in \mathbf{L}_\alpha[x]$ that $c^T_{i,\beta+1}(x) \in \mathbf{L}_\alpha[x]$ using Lemma 4.17 and the fact that $\alpha > \omega$. So at all stages of the computations, the content of the tape is inside $\mathbf{L}_\beta[x]$, so we have $\mathbf{space}(x,T) \leq \beta$, and hence $\mathbf{space}(x,T) \leq \mathbf{time}(x,T)$.

If $\alpha \leq \omega$ and $\alpha = \mathbf{time}(x,T)$, we have for all $\xi < \alpha$, by Theorem 5.5, $c^T_{i,\xi}(x) \in \mathbf{L}_\omega[x]$ for $i \in \{1,2,3\}$, and $\mathbf{space}(x,T) \leq \omega$ now follows directly. □

5.2 The question whether $\mathbf{P}_f = \mathbf{PSPACE}_f$

Now we have seen that, in all cases where the range of f contains only infinite ordinals, $\mathbf{P}_f$ is a subset of $\mathbf{PSPACE}_f$, one is inclined to wonder whether this inclusion can be shown to be proper. In this section, we will show that this, indeed, is the case at least for a number of functions f.

5.2.1 $\mathbf{P}_\alpha \neq \mathbf{PSPACE}_\alpha$ for ordinals up to ω^2

For the first result, the idea will be to construct a function that can be shown to be $\mathbf{PSPACE}_{\omega+2}$, but that, at the same time, can also be shown to be *not* $\mathbf{P}_\alpha$ for any $\alpha < \omega^2$. This will be done using the notion of arithmetical reals, which are defined more-or-less analogously to arithmetic sets of reals.

First, we observe the following fact about the location of recursive sets in Gödel's constructible hierarchy:

Lemma 5.8. *For any recursive set $w \in \mathbb{R}$, we can write w on the tape in ω steps, and thus we have that $w \in \mathbf{L}_{\omega+1}[\emptyset]$, and thus also $w \in \mathbf{L}_{\omega+1}[x]$ for any $x \in \mathbb{R}$.*

Proof. Because w is recursive, there is a Turing machine T that halts on all inputs $n \in \mathbb{N}$, such that $\phi_T(n) = 1$ if and only if $n \in w$. Now consider an ITTM that, in turn, for each number n, simulates the machine T, and then writes the correct number to the nth cell of the output tape. Because every individual computation will terminate in a finite number of steps, this can be done in ω steps, and we can go directly from the limit state to the halting state. □

We will now consider the notion of arithmetical sets of natural numbers: a set $S \subseteq \mathbb{N}$ is arithmetical if and only if there is a formula ϕ of PA such that $\phi(x) \iff x \in N$.

Lemma 5.9. *The set $A := \{x \in \mathbb{R} : x \text{ is arithmetical}\}$, is not arithmetical.*

Proof. This is shown in Example 13.1.9 in [Co]. □

This gives us the following result:

Lemma 5.10. *For any ordinal $\alpha \leq \omega^2$, the set*

$$A := \{x \in \mathbb{R} : x \text{ is arithmetical}\}$$

is not in $\mathbf{P}_\alpha$.

Proof. By Theorem 2.6 in [HaLe], the arithmetic sets are exactly the sets which can be decided in time $\omega \cdot n$ for some $n \in \mathbb{N}$. Because A is not arithmetical, it cannot be decided by any algorithm using a bounded finite number of limits, and hence, it is not in $\mathbf{P}_\alpha$ for any $\alpha \leq \omega^2$. □

It turns out, however, that this set A is in $\mathbf{PSPACE}_{\omega+1}$:

Theorem 5.11. *The set*

$$A := \{x \in \mathbb{R} : x \text{ is arithmetical}\}$$

is in $\mathbf{PSPACE}_{\omega+2}$.

Proof. First, we can note that we can, without any problems, enumerate all possible formulae that determine arithmetic sets, on one of the scratch tapes: every formula can be coded by a set which is finite, and hence in $\mathbf{PSPACE}_\omega$. Now, we may also assume that this enumeration only gives formulae in a normal form, with all quantifiers at the front.

Given a formula ϕ, we can also determine for any x whether $\phi(x)$ holds, by only writing down finite sets on a scratch tape. That this is the way can be shown by induction: if ϕ is quantifier free, we can simply evaluate the formula; if ϕ has an existential quantifier at the front and is of the form $\exists x_1 \psi$, we can write down every possible value n for x_1 at the front of the scratch tape, recursively evaluate $\psi[n/x_1]$ (which is possible by the inductive assumption) using the rest of the scratch tape; succeed if this is successful, and fail otherwise; if ϕ has a universal quantifier at the front and is of the form $\forall x_1 \phi$, we again write down every possible value n for x_1 at the front of the scratch tape, recursively evaluate $\psi[n/x_1]$, and now fail if this fails, and continue otherwise.

Now, the strategy for the whole function will be:

1. enumerate over all possible formulae in the aforementioned normal form;
2. for each such formula, test for each x whether the formula holds for x if and only if the xth position on the input tape is a 1;
3. succeed if we have found such a formula, and fail otherwise.

This is all possible while writing down only recursive sets. □

This gives us the following result on **P** and **PSPACE**:

Theorem 5.12. *For any α such that $\alpha \geq \omega + 2$ and $\alpha \leq \omega^2$, we have that $\mathbf{P}_\alpha \subsetneq \mathbf{PSPACE}_\alpha$, and even stronger, we have that $\mathbf{P}_\alpha \subsetneq \mathbf{PSPACE}_{\omega+2}$.*

Proof. This follows directly from the earlier results. □

Now we have shown that, for a certain number of α that are infinite, but still very low in the hierarchy of ordinals, the inclusion $\mathbf{P}_\alpha \subseteq \mathbf{PSPACE}_\alpha$ is a strict one. We might now wonder if we can also show a similar thing for any α larger than ω^2. It appears that this indeed is the case for recursive successor ordinals α. The strategy in showing this will be to show, that, for all recursive ordinals, $h_\alpha \in \mathbf{PSPACE}_{\alpha+1}$ which, together with the already known fact that $h_\alpha \notin \mathbf{P}_{\alpha+1}$, gives the desired result.

5.2.2 The case of recursive ordinals

We will now show that the inequality $\mathbf{P}_\alpha \neq \mathbf{PSPACE}_\alpha$ holds for much wider range of ordinals. From Proposition 4.9, we know that for all α, $h_\alpha \notin \mathbf{P}_{\alpha+1}$. It turns out, however, that for all recursive ordinals α, we do have that $h_\alpha \in \mathbf{PSPACE}_{\alpha+1}$:

Proposition 5.13. *For any recursive ordinal α, such that $\alpha \geq \omega + 1$, we have $h_\alpha \in \mathbf{PSPACE}_{\alpha+1}$.*

Proof. We can compute h_α in the following way, by making use of an ITTM_0 with several scratch tapes: to start, we will check whether the input corresponds to a natural number; if it does not, then we output 0, and if it does, we continue. Then, we will look if this natural number corresponds to a coding of an ITTM_0—again, we output 0 if this is not the case, and if it is, we continue. Note that the computation so far can be performed by only writing finite sets on the scratch tapes.

Now the real work can begin. First we write down the ordinal α, on the first scratch tape. Because α is recursive, we can do this in ω steps, so we are sure that the content of the first scratch tape, at all times, will be inside $\mathbf{L}_{\omega+1}[x]$ and, because $\alpha \geq \omega + 1$, also inside $\mathbf{L}_\alpha[x]$.

Once this is done, we will check for all ordinals β smaller than α (which can be easily found simply by restricting the ordinal written on the tape to all the elements smaller than a certain element, without affecting the space complexity in any way), whether the eth ITTM_0 has reached the halt state by stage β. If it turns out that this is the case, we look at the output: if the output is 0, then we will finish the computation by writing 1 on the output tape, and otherwise we finish by writing 0 on the output tape. If we reach stage β during the simulation without having halted, we go on checking with the next ordinal. Finally, if we have exhausted all ordinals $\beta < \alpha$, we again finish by writing 0 on the output tape.

In the case where α is a limit ordinal, we are now done, because we know that no machine can halt at any limit ordinal stage. In the case where α is a successor ordinal $\eta + 1$, however, we will additionally check whether the computation has finished at α itself.

For any $\beta < \alpha$, we know that $c^T_{i,\beta}[x] \in \mathbf{L}_\alpha[x]$ for $i \in \{1, 2, 3\}$. Furthermore, in the case where α is a successor ordinal $\eta + 1$, we know that $c^T_{i,\eta}[x]$ and $c^T_{i,\alpha}[x]$

can only differ by one element at most, and because α and η are known to be infinite, we obtain $\mathbf{L}_\alpha[x]$ from $\mathbf{L}_\eta[x]$ using Lemma 4.17.

It follows that this computation never writes a set on any of the tapes that is not in $\mathbf{L}_\alpha[x]$. This proves that $h_\alpha \in \mathbf{PSPACE}_{\alpha+1}$ indeed holds. □

Hence we have:

Theorem 5.14. *For every recursive successor ordinal $\alpha \geq \omega + 1$, $\mathbf{P}_\alpha \subsetneq \mathbf{PSPACE}_\alpha$ holds.*

Unfortunately, however, this process cannot be easily extended to work for limit ordinals: given a limit ordinal α, we can still compute h_α by only writing out information of space complexity less than α on tape, but it seems hard, if not impossible, to do this bounded by a specific ordinal β below α.

5.2.3 The case of clockable ordinals

It is, however, possible, to extend the above process to many writable successor ordinals. The strategy here is essentially the same as in the case of recursive ordinals: we know that h_α cannot be in $\mathbf{P}_{\alpha+1}$, and then we show that $h_\alpha \in \mathbf{PSPACE}_{\alpha+1}$. In the case of clockable ordinals, we can make use of the following theorem due to Philip Welch (Lemma 15 in [DeHaSc]):

Theorem 5.15. *If α is a clockable ordinal, then every ordinal up to the next admissible beyond α is writable in time $\alpha + \omega$.*

Besides this, we already know that $\alpha \in \mathbf{L}_{\alpha+1}[0]$. However, we will also have $\alpha \cup \{\{\{\{\emptyset\}\}, \eta\} : \eta \leq \beta\} \in \mathbf{L}_{\alpha+1}[0]$ for any $\beta \leq \omega$. Because these sets $\{\{\{\emptyset\}\}, \eta\}$ are not ordinals, it is immediate that α and $\{\{\{\{\emptyset\}\}, \eta\} : \eta \leq \beta\}$ are always disjoint. Thus, we can consider these sets $\alpha \cup \{\{\{\{\emptyset\}\}, \eta\} : \eta \leq \beta\}$ as alternative representations for the ordinals up to $\alpha+\omega$, that are within $\mathbf{L}_{\alpha+1}[0]$. This way, using this 'alternative' representation of the ordinals between α and $\alpha+\omega$, instead of the regular ones, in combination with the fact that the snapshots $c^T_{i,\alpha+n}[x]$ can only differ from $c^T_{i,\alpha}[x]$ by finitely many elements, we can represent computations of length $\alpha+\omega$ from x within $\mathbf{L}_{\alpha+2}[x]$ using a construction similar to that in section 5.1.

This gives us that, for clockable α, every ordinal up to the next admissible ordinal beyond α can be written on the tape with the content of the tape inside $\mathbf{L}_{\alpha+2}[0]$ at all stages. This gives us the following theorem:

Theorem 5.16. *If β is a clockable ordinal, and α is a successor ordinal between $\beta + 3$ and the next admissible after β, then we have $\mathbf{P}_\alpha \subsetneq \mathbf{PSPACE}_\alpha$.*

Proof. This goes largely analogous to the case of Proposition 5.13, with the major difference that we can now write α on the tape while staying inside $\mathbf{L}_{\beta+2}[0]$. As a result from this, if $\alpha = \eta + 1$, we get from $\eta \geq \beta + 2$ that $h_\eta \in \mathbf{PSPACE}_{\eta+1}$, whereas $h_\eta \notin \mathbf{P}_{\eta+1}$, giving the desired result. □

5.2.4 The case of suitable functions f

We can extend the above results from ordinals to suitable functions as defined in [DeHaSc]. There, a suitable function is defined as follows:

Definition 5.17. A function or a function-like operation f from $\mathbb{R}$ to the ordinals is called *suitable* whenever, for all reals x and y, $x \leq_{\mathrm{T}} y$ implies $f(x) \leq f(y)$, and if we have, for all x, $f(x) \geq \omega + 1$. The symbol $\leq_{\mathrm{T}}$ here stands for ordinary Turing reducibility as defined in e.g. [Co].

It turns out that most of the 'special' functions and operations that we have considered in this thesis are in fact suitable:

Proposition 5.18. *For any ordinal $\alpha > \omega + 1$, the constant function $f_0(x) = \alpha$ is suitable. Furthermore, the functions $f_1(x) = \omega_1^x$, $f_2(x) = \omega_1^x + \omega + 1$, $f_3(x) = \lambda^x$, $f_4(x) = \zeta^x$, and $f_5(x) = \Sigma^x$ are all suitable.*

Proof. For constant functions, we have $f_0(x) = f_0(y)$, and hence $f_0(x) \leq f_0(y)$ in all cases, so also in the specific case where $x \leq_{\mathrm{T}} y$. Hence, all constant functions are suitable.

For the function $f_1(x) = \omega_1^x$, assume that $x \leq_{\mathrm{T}} y$, and assume that, for some ordinal α, $\alpha < \omega_1^x$. This means that α is a x-recursive ordinal, and hence, that there is some real a coding α, such that $a \leq_{\mathrm{T}} x$. Now $a \leq_{\mathrm{T}} y$ follows by transitivity of $\leq_{\mathrm{T}}$, and thus α is also an y-recursive ordinal, and hence, $\alpha < \omega_1^y$. From this, $\omega_1^x \leq \omega_1^y$ directly follows. The case of $f_2(x) = \omega_1^x + \omega + 1$ follows directly, because $\alpha \leq \beta$ implies $\alpha + \omega + 1 \leq \beta + \omega + 1$.

For the function $f_3(x) = \lambda^x$, assume that $x \leq_{\mathrm{T}} y$, and assume that $\alpha < \lambda^x$, or, in other words, that α is a x-writable ordinal. We can now write α from y, by first computing x from y, and then going on with the computation that writes α from x. Hence, $\alpha < \lambda^y$, and $f_3(x) \leq f_3(y)$ follows directly. The cases of the functions ζ^x and Σ^x go nearly identically. □

We have the following results about suitable functions:

Theorem 5.19. *For any suitable function f and any set A of natural numbers,*

(i) $A \in \mathbf{P}_f$ if and only if $A \in \mathbf{P}_{f(0)+1}$

(ii) $A \in \mathbf{PSPACE}_f$ if and only if $A \in \mathbf{PSPACE}_{f(0)+1}$

Proof. (i) was originally proven in [DeHaSc, Theorem 26]; here we will provide a slightly modified version of the proof. Because for any natural number n, we have $0 =_{\mathrm{T}} n$, we get $f(0) = f(n)$ by the assumption of suitability; also, we have $0 =_{\mathrm{T}} \mathbb{N}$, so $f(0) = f(\mathbb{N})$; moreover, for any real number x, we have $0 \leq_T x$, and hence $f(0) \leq f(x)$. Now consider the constant function g such that $g(x) = f(0)$ for all x. As a direct result of the definition, we obtain $\mathbf{P}_{f(0)+1} = \mathbf{P}_g$. We also have $g(x) \leq f(x)$ for all x, so the result $\mathbf{P}_g \subseteq \mathbf{P}_f$ is immediate.

For the converse, assume that $A \in \mathbf{P}_f$, and that T is a time f machine deciding A. We now construct a time g machine T', which performs the same computation as T, while, during the first ω steps of the computation, simultaneously checking if the input actually codes a natural number or the entire set $\mathbb{N}$. Because a natural number n, when considered as a real, is equal to the set $\{0, \ldots, n-1\}$, and any such set corresponds to a natural number n, it follows that a real x does *not* correspond to a natural number or the complete set $\mathbb{N}$ if and only if the string 01 occurs in it. We now ensure that T', during the first ω steps, searches for the string 01, and halts whenever this string is encountered, while simultaneously simulating T. After we first reach the limit state, we know

that the input did not contain the string 01, and we continue the original computation of T. If no 01 is encountered, the input x must be either a natural number n, or the complete set of natural numbers $\mathbb{N}$, and it will finish within time $f(x) = f(0)$. If a 01 is encountered in the input x, it is encountered within the first ω steps, and hence $f(x) < \omega < f(0)$. So T' is a time $f(0)$-machine deciding A, and hence, $A \in \mathbf{P}_{f(0)+1}$.

(ii) can be proven similarly. It again follows directly that $\mathbf{PSPACE}_{f(0)+1} \subseteq \mathbf{PSPACE}_f$. The converse now is a bit simpler. If T is a space f machine deciding A, consider the following machine T': on input x, we first check, without making any modifications (and thus, while staying within $\mathbf{L}_0[x]$), whether x is a natural number. We output 0 if it does not, and if it does, we continue the computation of which we now know that $\mathbf{space}(x, T) < f(x) = f(0)$. Because at the start of this computation, the tape is still unchanged, and the algorithm performed after the check if x is a natural number, is identical, we also obtain $\mathbf{space}(x, T) < f(0)$. It is now clear that $\mathbf{space}(x, T') < f(0)$ for all x, so T' is a space $f(0)$-machine deciding A, so $A \in \mathbf{PSPACE}_{f(0)+1}$. □

Now, because for any α, the set h_α always consists of only natural numbers, we can directly extend the earlier results about ordinals to results about suitable functions f:

Theorem 5.20. *If f is a suitable function, and $f(0)$ is either recursive, or a clockable ordinal, such that there is an ordinal β such that $f(0)$ is between $\beta+2$ and the next admissible ordinal after β, then we have $\mathbf{P}_f \subsetneq \mathbf{PSPACE}_f$.*

Proof. On one hand, we have $h_{f(0)+1} \notin \mathbf{P}_{f(0)+2}$ from Proposition 4.9, which, by Theorem 5.19 gives us $h_{f(0)+1} \notin \mathbf{P}_{f+1}$. On the other hand, we have $h_{f(0)+1} \in \mathbf{PSPACE}_{f(0)+2}$ from either Proposition 5.13 or Proposition 5.16, which gives us $h_{f(0)+1} \in \mathbf{PSPACE}_{f+1}$. Hence we have $\mathbf{P}_f \neq \mathbf{PSPACE}_f$, and the result follows. □

5.2.5 However, $\mathbf{P}_f = \mathbf{PSPACE}_f$ for 'almost all' f

So far, we have shown that, for certain classes of 'low' functions f and ordinals α, there is a strict inclusion $\mathbf{P}_f \subset \mathbf{PSPACE}_f$. This brings us to wonder we can also find functions and ordinals where this strict inclusion does not hold, and instead we have an equality $\mathbf{P}_f = \mathbf{PSPACE}_f$.

It is easy to see that we will have this equality, at least for very high, non-countable, ordinals and functions: if we have $\alpha > \omega_1$, then we must have $\mathbf{P}_\alpha = \mathbf{Dec}$, and because we also have $\mathbf{PSPACE}_\alpha \subseteq \mathbf{Dec}$ and $\mathbf{P}_\alpha \subseteq \mathbf{PSPACE}_\alpha$, we indeed obtain $\mathbf{P}_\alpha = \mathbf{PSPACE}_\alpha$. However, we can generalize this towards a wider range of functions:

Proposition 5.21. *If f satisfies $f(x) \geq \lambda^x$ for all x, then we have $\mathbf{P}_f = \mathbf{PSPACE}_f$.*

Proof. We clearly have $\mathbf{P}_f = \mathbf{Dec}$ because f is, on all x, larger than the supremum of halting times of ITTM computable functions on input x, as a result of the fact that $\gamma^x = \lambda^x$ for all x (Proposition 3.18). Also, we have $\mathbf{P}_f \subseteq \mathbf{PSPACE}_f$, as well as $\mathbf{PSPACE}_f \subseteq \mathbf{Dec}$. Hence, we have $\mathbf{P}_f = \mathbf{Dec} = \mathbf{PSPACE}_f$. □

As this class of functions f is a superset of the class of functions f—called 'almost all f' there—for which it is shown, in [HaWe], that $\mathbf{P}_f \neq \mathbf{NP}_f$, we can, with a little wink, indeed say that we have $\mathbf{P}_f = \mathbf{PSPACE}_f$ for 'almost all' f.

Chapter 6

Koepke machines

So far, we have only looked at extensions of Turing machines into ordinal time: in the earlier defined Hamkins-Kidder machines, the length of the tape, however, has remained of size ω. In this chapter, we will consider a different approach towards infinitary computation than the one outlined in the earlier chapters, which has been outlined by Peter Koepke in [Ko]. There, so-called 'ordinal Turing machines' are defined, which not only are able to have computation lengths of (in principle) any ordinal length, but also have a class-sized tape with the size of the class of all ordinals.

As it turns out, these machines are strictly more powerful than the earlier Hamkins-Kidder model of infinite time Turing machines: it has been shown in [Lö] that the Hamkins-Kidder weak halting problem h is decidable by one of these Ordinal machines.

In this chapter, we will look at the analogues of the problems about time and space complexity classes presented in earlier chapters. Although in principle, here it is possible to look at classes of decidable ordinals (which need not even be sets!), in this thesis, the **P** and **PSPACE** classes will be defined as containing only sets of reals decidable within a certain time and space complexity.

From this point onward, we will generally refer to the earlier classes **P**, **PSPACE** and the likes, defined with respect to Hamkins-Kidder computations, as $\mathbf{P}^{\mathrm{HK}}$ and $\mathbf{PSPACE}^{\mathrm{HK}}$ respectively, whereas we will call the classes relating to Koepke's 'ordinal Turing machine' as $\mathbf{P}^{\mathrm{K}}$ and $\mathbf{PSPACE}^{\mathrm{K}}$. Similarly, we will refer to the earlier defined weak halting problem h as h^{HK} from now on.

6.1 Definitions

The definition of Koepke machines will be largely similar to that of Hamkins-Kidder machines of the ITTM_1-type; the only major difference being in the length of the tapes:

Definition 6.1. An *ordinal Turing machine* or a *Koepke machine* is, like the ITTM_1 machines, defined by a tuple of the form

$$T = (Q, \delta, q_s, q_f)$$

where Q is a set of internal states, δ is a transition function from $(Q \backslash \{q_f\} \times$

$\{0,1\}^3$ to $Q \times \{0,1\}^3 \times \{L,R\}$ and $q_s, q_f \in Q$ are two special states called the initial and halting state, such that $q_s \neq q_f$ holds.

In the case of ordinal Turing machines, we will again use the same notations $q^T_\beta(w)$, $h^T_\beta(w)$, $c^T_{i,\beta}(w)$, and $\phi_T(w) = v$ as in Definition 2.2.

The computation, too, will go largely analogous to that of Hamkins-Kidder machines as outlined in Definition 2.1, with the following exceptions:

1. If a machine, during a computation, is ordered to move 'left' while the head is at a limit ordinal, it moves to position 0. Formally: if $h^T_\alpha(x) = \beta$ where β is a limit ordinal, and the third component of $\delta(q^T_\alpha, (c^T_{1,\beta,\alpha}, c^T_{2,\beta,\alpha}, c^T_{3,\beta,\alpha}))$ is L, then $h^T_{\alpha+1}(x) = 0$.

2. At limit stages of the computation, the position of the head will, instead of 0, be set to the lim inf of the earlier stages visited by the head. Formally: if α is a limit ordinal, then $h^T_\alpha(x) = \liminf\{h^T_\beta(x) : \beta < \alpha\}$.

3. Like in the ITTM$_1$-type Hamkins-Kidder machines, at limit stages, the state will be equal to the lim sup of the earlier visited states. Formally: $q^T_\alpha(x) = \limsup\{q^T_\beta(x) : \beta < \alpha\}$.

As a result of defining the position of the head at limit stages as the lim inf of the earlier head positions, it follows directly that the head position may, at some point, be an infinite ordinal. This, in turn implies that infinite ordinals may be written to, or again deleted from the tapes: as a result, for an arbitrary ordinal α, any $x \in \mathbb{R}$, and any $i \in \{1,2,3\}$, $c^T_{i,\alpha}(x)$ need not be a real anymore, but can in principle be any set of ordinals. In principle, we could also remove the restriction that the input of a function be a real, and replace it with the weaker requirement that it be a set of ordinals. However, in this thesis we will not do this, and focus only at computations by Koepke machines starting from reals.

Now we have defined the Koepke machines and their computation, we continue by defining the complexity classes **P** and **PSPACE**. In the case of **P**, the definitions are largely identical to those for Hamkins-Kidder machines:

Definition 6.2. If T is a machine that eventually reaches the halting state q_f, and α is the smallest ordinal such that $q^T_\alpha(x) = q_\mathrm{f}$, then we say that $\mathbf{time}(x, T) = \alpha$.

Definition 6.3. For any Koepke machine T, we say that T is a **time f machine** if, for all $x \in \mathbb{R}$, we have $\mathbf{time}(x, T) \leq f(x)$. For any ordinal ξ, we say that T is a **time ξ machine** if T is a time f machine for the constant function f such that $f(x) = \xi$ for all $x \in \mathbb{R}$.

Using these definitions, we can define the classes $\mathbf{P}^\mathrm{K}_f$ and $\mathbf{P}^\mathrm{K}_\alpha$ for functions f and ordinals α, just like we did the same thing earlier for Hamkins-Kidder machines.

Definition 6.4. For any function f, we let $\mathbf{P}^\mathrm{K}_f$ denote the class of all sets of reals that are decidable by a time f machine. For any ordinal ξ, we let $\mathbf{P}^\mathrm{K}_\xi$ denote the class of all sets of reals that are decidable by a time η machine for some $\eta < \xi$.

For the space complexity classes, we are now able to use a much simpler definition than in the case of Hamkins-Kidder machines: we can simply define $\mathbf{space}(x, T)$ as the largest α that occurs as the position of the head at some stage of the computation:

Definition 6.5. If T is a machine that, at one point reaches the halting state q_{f}, and $\mathbf{time}(x, T) = \alpha$, we define $\mathbf{space}(x, T) = \sup\{h^T_\beta(x) : \beta \leq \alpha\}$.

Definition 6.6. For any infinite time Turing machine T, we say that T is a **space f machine** if, for all $x \in \mathbb{R}$, we have $\mathbf{space}(x, T) \leq f(x)$. For any ordinal ξ, we say that T is a **space ξ machine** if T is a space f machine for the constant function f such that $f(x) = \xi$ for all $x \in \mathbb{R}$.

Definition 6.7. For any function f, we let $\mathbf{PSPACE}^{\mathrm{K}}_f$ denote the class of all sets of reals that are decidable by a space f machine. For any ordinal ξ, we let $\mathbf{PSPACE}^{\mathrm{K}}_\xi$ denote the class of all sets of reals that are decidable by a space η machine for some $\eta < \xi$.

An immediate result of these definitions, is that a computation of a Koepke machine can never take more space than it can take time:

Proposition 6.8. *For all f and all α, we have $\mathbf{P}^{\mathrm{K}}_f \subseteq \mathbf{PSPACE}^{\mathrm{K}}_f$ and $\mathbf{P}^{\mathrm{K}}_\alpha \subseteq \mathbf{PSPACE}^{\mathrm{K}}_\alpha$ respectively.*

Proof. It is easy to see by ordinal induction on α, that we have $h^T_\alpha(x) \leq \alpha$ for all x and all α. From this, it follows immediately from the definitions that $\mathbf{space}(x, T) \leq \mathbf{time}(x, T)$. Hence, for any ordinal α, we have $A \in \mathbf{PSPACE}^{\mathrm{K}}_\alpha$ if $A \in \mathbf{P}^{\mathrm{K}}_\alpha$, and for any function f, we have $A \in \mathbf{PSPACE}^{\mathrm{K}}_f$ if $A \in \mathbf{P}^{\mathrm{K}}_f$. □

6.2 Variants on the Koepke machine architecture

Like in the case of Hamkins-Kidder machines, we can again consider variations on the standard Koepke machine model. Although we will not define an analogue of ITTM_0 machines here, we will briefly mention Koepke machines with n tapes for $n \neq 3$ and Koepke machines with stay option.

Proposition 6.9. *For any m and any n, a set A is decidable by a Koepke machine with m tapes and n scratch cells if and only if it is decidable by a Koepke machine with m tapes and n scratch cells and a stay option.*

Proof. Proposition 2.7 can be applied after making exactly the same modifications that were made in Proposition 2.9. □

Proposition 6.10. *For any $n \in \mathbb{N}$, with $n > 2$, a set A is decidable by a Koepke machine with n tapes if and only it is decidable by a Koepke machine with 2 tapes.*

Proof. Because we only are considering sets of reals here, we only have to deal with inputs that are real numbers. Assume that A is decidable by a Koepke machine T with n tapes. We can now construct a Koepke machine T' with 2 tapes as follows:

- T' starts out by stretching the input such that, if x is the input, after the stretching operation we find $\{(n-1)\cdot k : k \in x\}$ on the first tape. We again can use a signal state to recognize when we are done with this operation, and ensure this signal state only reads 1s at successor ordinals; at stage ω we will find ourselves with the head at position ω, in this signal state, and read a 0, and move left so that the head position will again be 0.

- After this, we simulate the original n tape machine on our 2 tape machine, using the second tape to simulate the output tape, and the first tape to simulate the $n-1$ other tapes. We make sure that we have a fixed number k such that each step by T is simulated by k steps of T', and moreover ensure that, for any ordinal α such that $\alpha = k \cdot \beta$ for some k, $h^T_\alpha(x) = (n-1)\cdot\eta$ for some η.

An important difference from the case of Hamkins-Kidder machines, is the head position at limit stages: by making sure that the least ordinal visited cofinally often will always be some ordinal α, such that $\alpha = k \cdot \beta$, we can ensure that T' will correctly simulate T.

Because the only tape being simulated on the second tape is the original second tape, and because first cell of the second tape of T corresponds to the first cell of the second tape of T', it follows that $\phi_T(x) = 1$ if and only if $\phi_{T'}(x) = 1$. □

This gives us the following corollary:

Corollary 6.11. *For any $m, n > 2$, a set A is decidable by a Koepke machine with m tapes if and only if it is decidable by a Koepke machine with n tapes.*

6.3 Time complexity for Koepke machines

We can prove a number of theorems about time complexity for Koepke machines: to start with, it turns out that for many recursive ordinals α, a Koepke machine can compute exactly the same sets in time α that a Hamkins-Kidder machine can:

Proposition 6.12. *For any recursive ordinal α larger than ω^2, where $\alpha = \beta+1$, and β is a limit ordinal, we have $\mathbf{P}^{\mathrm{HK}}_\alpha = \mathbf{P}^{\mathrm{K}}_\alpha$.*

Proof. Assume that $A \in \mathbf{P}^{\mathrm{K}}_\alpha$. This means that A is decidable by a time η machine for some $\eta < \alpha$, which, because $\alpha = \beta + 1$, here can be restated as: A is decidable by a time β Koepke machine. Then A is, by Proposition 6.8, decidable by a time β Koepke machine that uses β or less cells. Because β is a recursive ordinal, we can write a code for β on the tape in ω steps. After that, we can simulate the Koepke algorithm on a tape of size β on our ω-sized tape. Because every step of the simulated computation will be simulated by a finite number of steps of the simulating computation, and because the simulated computation takes less than β steps (the halting state cannot be reached at β itself, because β is a limit ordinal), the simulating computation will also take less than β steps. Because $\beta \geq \omega^2$, the complete algorithm will take less than $\omega + \beta = \beta$ steps in total. □

This equivalence between the time complexity classes of Koepke machines and Hamkins-Kidder machines also holds at the level λ, the supremum of all writable ordinals:

Proposition 6.13. $\mathbf{P}_\lambda^{\mathrm{HK}} = \mathbf{P}_\lambda^{\mathrm{K}}$.

Proof. Assume that A is decidable by a time α Koepke machine T, where $\alpha < \lambda$ or, in other words, α is a writable ordinal as a result of the fact that the class of writable ordinals is downward closed (Proposition 3.19). Clearly this machine T is also a space α machine, so the computation can use at most α cells. Say, α is writable by a Hamkins-Kidder machine in time β. Now β too must be a writable ordinal, due to the fact that it is clockable as there is a computation starting on input 0 finishing in time β, and the fact that every clockable ordinal is writable as a result of Corollary 3.20. After writing a code for α on the tape in β steps, we can continue by simulating the Koepke algorithm by simulating a tape of size α, in time less than α. Thus, we can decide A with a Hamkins-Kidder machine in time $\beta + \alpha$, which is again a writable ordinal by Proposition 3.22, and hence smaller than λ. □

It turns out, that in fact, an equality of the above type still holds up to the level Σ^x, the supremum of all ordinals accidentally writable on input x:

Proposition 6.14. $\mathbf{P}_{\Sigma^x}^{\mathrm{K}} = \mathbf{P}_{\lambda^x}^{\mathrm{HK}}$.

Proof. Assume that a set A is in $\mathbf{P}_{\Sigma^x}^{\mathrm{K}}$. Then it follows from Proposition 6.8 that A is in $\mathbf{PSPACE}_{\Sigma^x}^{\mathrm{K}}$, and we know that there is an accidentally x-writable ordinal α such that, if we can write α on the tape, we can compute A on a Hamkins-Kidder machine, simulating an α-sized fragment of the tape of the Koepke machine within space ω, based upon the just-found coding of α.

The tactic now is to construct a Hamkins-Kidder machine, that simulates all other Hamkins-Kidder machines on input x, and in doing so, performs an algorithm that, one by one, writes all reals that are accidentally writable from x on one of the scratch tapes. We can be sure that every real that is accidentally writable from x will be written on one of the scratch tapes at some point; and, each time a real is written on the scratch tape this way, we check if it codes an ordinal. If it does, we continue simulating the computation using this ordinal as tape length: eventually we will either run out of space, in which we continue the algorithm writing accidentally writable reals on the tape, or it will turn out that we have sufficient space, in which case the computation will finish.

This computation will eventually halt, because a real coding an ordinal equal to or larger than α will be written on the tape at some point and, as all halting computations will halt before λ^x, we know that the computation will halt before λ^x. So, it follows that the set A is in $\mathbf{P}_{\lambda^x}^{\mathrm{HK}}$. □

This gives us the following corollary:

Corollary 6.15. $\mathbf{P}_{\Sigma^x}^{\mathrm{K}} = \mathbf{Dec}^{\mathrm{HK}}$

Proof. We have $\mathbf{P}_{\Sigma^x}^{\mathrm{K}} \subseteq \mathbf{Dec}^{\mathrm{HK}}$ as a result of $\mathbf{Dec}^{\mathrm{HK}} = \mathbf{P}_{\lambda^x}^{\mathrm{HK}}$ and $\mathbf{P}_{\lambda^x}^{\mathrm{HK}} = \mathbf{Dec}^{\mathrm{HK}}$ (the latter equality holds because $\lambda^x = \gamma^x$, so all halting computations from x must halt before λ^x). □

6.4 Space complexity for Koepke machines

Now we will take a quick look look at space complexity issues for Koepke-type Ordinal machines.

Proposition 6.16. $\mathbf{Dec}^{\mathrm{HK}} \subseteq \mathbf{PSPACE}^{\mathrm{K}}_{\omega+2}$

Proof. Assume that $A \in \mathbf{Dec}^{\mathrm{HK}}$. We may assume, as a result of the equivalences shown in section 2.5, that A is decided by an ITTM_1 machine that is able to recognize when it is at a limit stage. We now can simulate A on a Koepke machine with tape length $\omega + 1$ by moving left ω times[1] at any limit state, and besides this, simply computing the same program that the Hamkins-Kidder machine uses. □

Proposition 6.17. $\mathbf{PSPACE}^{\mathrm{K}}_{\Sigma^x} \subseteq \mathbf{Dec}^{\mathrm{HK}}$

Proof. In the proof Proposition 6.14, it was shown that $\mathbf{PSPACE}^{\mathrm{K}}_{\Sigma^x} \subseteq \mathbf{P}^{\mathrm{HK}}_{\lambda^x}$. In combination with the fact that $\mathbf{P}^{\mathrm{HK}}_{\lambda^x} = \mathbf{Dec}^{\mathrm{HK}}$, this gives the desired result. □

These two results combined give an interesting result: together they imply that it does not really matter how much extra space you use on an ordinal Turing machine, as long as it does not go beyond Σ^x. In particular, we have:

Corollary 6.18. *For any ordinal α, such that $\alpha \geq \omega + 1$ and $\alpha \leq \Sigma + 1$, and for any function f such that we have $f(x) \geq \omega$ and $f(x) \leq \Sigma^x$ for all x, we have respectively that $\mathbf{PSPACE}^{\mathrm{K}}_{\alpha} = \mathbf{Dec}^{\mathrm{HK}}$ or $\mathbf{PSPACE}^{\mathrm{K}}_{f} = \mathbf{Dec}^{\mathrm{HK}}$.*

So, it turns out that, even with space Σ^x at our disposal, we are unable to compute any functions that a Hamkins-Kidder machine cannot compute. Moreover, we can now establish a relationship between the $\mathbf{PSPACE}^{\mathrm{K}}$-classes and the $\mathbf{P}^{\mathrm{K}}$-classes in a few cases.

Proposition 6.19. *For any ordinal α, such that $\alpha \geq \omega+1$ and $\alpha \leq \lambda$, we have that $\mathbf{P}^{\mathrm{K}}_{\alpha} \subsetneq \mathbf{PSPACE}^{\mathrm{K}}_{\alpha}$.*

Proof. On one side, we know from Corollary 6.18 that $\mathbf{PSPACE}^{\mathrm{K}}_{\alpha} = \mathbf{Dec}^{\mathrm{HK}}$, and on the other side, we know that $\mathbf{P}^{\mathrm{K}}_{\alpha} \subsetneq \mathbf{Dec}^{\mathrm{HK}}$. □

Proposition 6.20. *For functions f such that, for all x, $\Sigma^x > f(x) > \lambda^x$, we have $\mathbf{PSPACE}^{\mathrm{K}}_{f} = \mathbf{P}^{\mathrm{K}}_{f} = \mathbf{Dec}^{\mathrm{HK}}$.*

Proof. On one side, from Corollary 6.18 it follows that $\mathbf{PSPACE}^{\mathrm{K}}_{f} = \mathbf{Dec}^{\mathrm{HK}}$. On the other side, we know that $\mathbf{P}^{\mathrm{K}}_{f} \supseteq \mathbf{P}^{\mathrm{K}}_{\lambda^x}$, and from Proposition 6.14 we know that $\mathbf{P}^{\mathrm{K}}_{\lambda^x} = \mathbf{P}^{\mathrm{HK}}_{\lambda^x}$. From the fact that $\mathbf{P}^{\mathrm{HK}}_{\lambda^x} = \mathbf{Dec}^{\mathrm{HK}}$, it thus follows that $\mathbf{P}^{\mathrm{K}}_{f} \supseteq \mathbf{Dec}^{\mathrm{HK}}$. So we know that $\mathbf{PSPACE}^{\mathrm{K}}_{f} = \mathbf{Dec}^{\mathrm{HK}}$, as well as that $\mathbf{P}^{\mathrm{K}}_{f} \supseteq \mathbf{Dec}^{\mathrm{HK}}$. Finally, from Proposition 6.8 we know that $\mathbf{P}^{\mathrm{K}}_{f} \subseteq \mathbf{PSPACE}^{\mathrm{K}}_{f}$, so that we must have $\mathbf{P}^{\mathrm{K}}_{f} = \mathbf{Dec}^{\mathrm{HK}}$, completing the proof. □

[1] If, at a limit state, the head is at position $n < \omega$, we need to move left n times in order to make sure that the new head position is 0; and if the head is at position ω itself, we need to move left only once; so, by ensuring that the machine gets the instruction to move left ω times in a row, we ensure ourselves that we end up with the head at position 0, as desired, regardless of the original tape position.

At the same time, we will prove in Proposition 6.21 that Koepke machines *are* in fact more powerful than Hamkins-Kidder machines, even when it comes to computing sets of reals. To see this, we will let ourselves be guided by the following question: are there any countable functions or ordinals, such that there are functions computable by a Koepke machine, bounded in space by respectively that ordinal or function, that are not decidable by any Hamkins-Kidder machine?

It turns out that this, indeed is the case. With Proposition 3.16, we can show that the weak halting problem h is indeed in $\mathbf{PSPACE}^{\mathrm{K}}_{\Sigma+2}$:

Proposition 6.21. $h \in \mathbf{PSPACE}^{\mathrm{K}}_{\Sigma+2}$.

Proof. We will sketch a possible way to compute h in $\mathbf{PSPACE}^{\mathrm{K}}_{\Sigma+2}$ space. First we will simply check if the input codes a natural number n; if it does not, we simply output 0.

If it does, we know that either the computation of Hamkins-Kidder machine n on input n will finish before time λ, or, as a result of Proposition 3.16 the contents of the tape at stage ζ will be identical to that at stage Σ.

Now, we will simulate the (non-halting) computation of h on input n, but, using a scratch tape, we will keep track of all the ω-sized tapes at any stage. If we start a new step in the computation, we will first check if the configuration at this step is equal to one at an earlier step. If this is the case, we halt. Because the tape of the simulated computation has size ω, at any time α, we will have used at most $\omega \cdot \alpha$ tape. This leaves us with the following two possibilities:

- The Hamkins-Kidder machine n, on input n, finishes before time λ. In this case, $n \in h$, and we output 1, and we have used less than $\omega \cdot \lambda < \Sigma$ space.
- The Hamkins-Kidder machine n, on input n, never finishes, and the content of stage Σ will be equal to that at time ζ due to Proposition 3.16. We will realize this during the checking stage at step Σ, and at that point the used part of the tape has size Σ.

In either case, we will use at most Σ space during the computation, and thus we have obtained a computation of h in $\mathbf{PSPACE}^{\mathrm{K}}_{\Sigma+2}$ space. □

The consequences of the above are essentially the following: up to the level of Σ^x, Koepke machines bounded in space up to level Σ^x cannot compute any sets that Hamkins-Kidder machines cannot compute. However, just above the level Σ this changes: in fact, the weak halting problem h is contained in $\mathbf{PSPACE}^{\mathrm{K}}_{\Sigma+2}$ as well as in $\mathbf{PSPACE}^{\mathrm{K}}_{f}$ with f the function such that $f(x) = \Sigma + 1$ for all $x \in \mathbb{N}$, and $f(x) = \omega + 1$ for all $x \in \mathbb{R}\backslash\mathbb{N}$.

Chapter 7

Nondeterministic computation

In this chapter we will look at the idea of nondeterministic computation, applied to Hamkins-Kidder machines as well as Koepke machines. Whereas the earlier part of this thesis was mostly focussed on several variations of the question $\mathbf{P} \stackrel{?}{=} \mathbf{PSPACE}$, and the question $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ has already been treated in earlier literature, in this chapter we will mostly focus on the questions $\mathbf{NP} \stackrel{?}{=} \mathbf{PSPACE}$ and $\mathbf{PSPACE} \stackrel{?}{=} \mathbf{NPSPACE}$ respectively. Because the definitions for space complexity differ so widely, the two cases of Hamkins-Kidder machines and Koepke machines have to be treated separately.

7.1 Nondeterministic computation in the case of Hamkins-Kidder machines

7.1.1 Time complexity

For the task of defining nondeterministic classes of computation, there appear to be two possible approaches to incorporate the notion of indeterminism: first, we can try to provide for the nondeterminism by taking letting machines take 'arbitrary' reals as extra argument in addition to the standard argument; and second, we can try to define a notion of a 'nondeterministic infinite time Turing machine', define computations by such machines, and base our notions of complexity classes on those definitions. In most of the earlier papers, such as [Sc], [DeHaSc], and [HaWe], the first approach was taken; in [Lö], on the other hand, the second approach was introduced. However, it turns out, and is proven in [Lö], that, in many of the important cases, the two different definitions are equivalent. Here, both definitions will be given, and the equivalence between the two will again be proven. The original **NP** class will here be called $\mathbf{NP}'$, whereas we will use the notation **NP** for the class based upon nondeterministic infinite time Turing machines.

Definition 7.1. For any function f and any set A, we say that $A \in \mathbf{NP}'_f$, if there is a time f machine T such that for any x, we have that $x \in A$ if and

only if there is a real y such that T accepts $x \oplus y$. Likewise, for any ordinal α and any set A, we say that $A \in \mathbf{NP}'_\alpha$, if there is a time η machine T for some $\eta < \alpha$, such that for any x, we have that $x \in A$ if and only if there is a real y such that T accepts $x \oplus y$.

Now let us continue with the second approach, and define nondeterministic infinite time Turing machines. The main modification from the standard ITTM_0 model that we need to make in order to allow nondeterministic computation, essentially comes down to allowing the transition function δ to be any relation, rather than a function, so that from any state and tape content, there may be different possible courses of action.

Definition 7.2. A **nondeterministic infinite time Turing machine** is defined like a regular ITTM, but here δ is a relation between $(Q \backslash q_\text{f}) \times \{0,1\}^3$ and $Q \times \{0,1\}^3 \times \{L, R\}$, rather than a function.

We need to take some care in defining the notion of a 'computation' for nondeterministic machines. A first attempt to come up with a definition would be, to say that a computation of a nondeterministic infinite time Turing machine is any function-like definite operation A such that:

$$\alpha \to (q_\alpha(x), h_\alpha(x), c_{1,\alpha}(x), c_{2,\alpha}(x), c_{3,\alpha}(x))$$

which is defined on all ordinals. This definition, however, is problematic, because A is necessarily a proper class, as it would contain an element for every ordinal. Moreover, we are not even guaranteed that any such A even exists as a class, because we do not have an explicit definition of A.

We, however, can define *partial* computations much easier: we say that any set A is a **partial computation** on a machine T from x, if A is a function:

$$\alpha \to (q_\alpha(x), h_\alpha(x), c_{1,\alpha}(x), c_{2,\alpha}(x), c_{3,\alpha}(x))$$

that has a downward closed set of ordinals as domain, and which satisfies the requirements of an infinite time Turing machine computation named in Definition 2.2. On any partial computation p by a nondeterministic infinite time Turing machine from x, we can now also define the operations q^p_α, h^p_α, and $c^p_{i,\alpha}$ for $i \in \{1,2,3\}$ as the current head, the head position, and the content of the tape respectively at stage α of the partial computation p, and we can define $\text{len}(p)$ as the largest ordinal in the domain of p. We call a partial computation a **halting computation**, if q_f occurs at some point in it.[1] If p is a halting computation by a machine T from input x, and if $\text{len}(p) = \alpha$, we furthermore write $\mathbf{time}(x, T, p) = \alpha$.

We now define the notion of **uniform halting** a nondeterministic computation by T from x: we say that a nondeterministic infinite time Turing machine T halts uniformly on input x, if: a) the class of all halting computations is a set A, and b) every partial computation computation on T has an extension in A. Furthermore, we say that a nondeterministic infinite time Turing machine T **uniformly halts in time** f, if it uniformly halts, and, on every halting computation by T from x, the state q_f is reached at, or before, $f(x)$.

[1] If q_f occurs anywhere in a partial computation, it must be necessarily at the last stage of it, as a result of the definition of δ as a relation between $(Q \backslash q_\text{f}) \times \{0,1\}^3$ and $Q \times \{0,1\}^3 \times \{L, R\}$

Definition 7.3. For any nondeterministic infinite time Turing machine T, we say that T is a **nondeterministic time f machine** if for all x, the computation of T on x uniformly halts at time $f(x)$. For any ordinal ξ, we say that T is a **nondeterministic time ξ machine** if T is a nondeterministic time f machine for the constant function f such that $f(x) = \xi$ for all x.

Definition 7.4. For any function f, we let $\mathbf{NP}_f$ denote the class of all sets of reals that are decidable by a nondeterministic time f machine. For any ordinal ξ, we let $\mathbf{NP}_\xi$ denote the class of all sets of reals that are decidable by a time η machine for some $\eta < \xi$.

The classes $\mathbf{NP}$, $\mathbf{NP}_+$, and $\mathbf{NP}_{++}$ are also defined analogously to the classes $\mathbf{P}$, $\mathbf{P}_+$, and $\mathbf{P}_{++}$. We also let $\mathbf{NDec}$ denote the class of all sets that are decidable by any nondeterministic infinite time Turing machine.

Finally, we will define classes that contain all the sets which have a complement in the $\mathbf{NP}$-classes:

Definition 7.5. For any function, f, we let $\mathbf{coNP}_f$ denote the class of all sets of reals A such that $A \in \mathbf{coNP}_f$ if and only if $\mathcal{P}(\mathbb{R})\backslash A \in \mathbf{NP}_f$. Similarly, for any ordinal α, we let $\mathbf{coNP}_\alpha$ denote the class of all sets of reals A such that $A \in \mathbf{coNP}_\alpha$ if and only if $\mathcal{P}(\mathbb{R})\backslash A \in \mathbf{NP}_\alpha$.

The equivalence, or at least equivalence for 'important' ordinals and functions, between the $\mathbf{NP}$-classes and the $\mathbf{NP}'$-classes is established by the following proofs, which appeared earlier as Proposition 11 in [Lö]:

Proposition 7.6. *Let f be a function from $\mathbb{R}$, the powerset of $\mathbb{N}$, to* Ord*, the class of ordinals, such that $f(x) \geq \omega^2$ for all x. Then, if $A \in \mathbf{NP}'_f$, we also have $A \in \mathbf{NP}_f$.*

Proof. Assume that $A \in \mathbf{NP}'_f$, and that for all x, $f(x) > \omega^2$. We now know that there is a time f machine T such that for any x, $x \in A$ if and only if there is a real y such that T accepts $x \oplus y$. We now can construct a nondeterministic machine that first, in ω steps, writes down a real nondeterministically on the even positions of the tape, and then copies the action of T. This is a nondeterministic Hamkins-Kidder machine that decides A in $\omega + f(x)$ steps. Because $f(x) \geq \omega^2$, $\omega + f(x)$ must be equal to $f(x)$, so it follows that $A \in \mathbf{NP}_f$. □

Proposition 7.7. *Let f be a function from $\mathbb{R}$, the powerset of $\mathbb{N}$, to* Ord*, the class of ordinals, such that $\omega \cdot 2 \cdot f(x) = f(x)$ for all x, and $f(x) < \omega_1$ for all x. Then, if $A \in \mathbf{NP}_f$, we also have $A \in \mathbf{NP}'_f$.*

Proof. Assume that $A \in \mathbf{NP}_f$. Then there is a nondeterministic Hamkins-Kidder machine T that, on any input x finishes before time $f(x)$ on every branch of the computation. Because $f(x)$ is a countable ordinal, there is some real coding $f(x)$ as a relation on the natural numbers. If we regard this real as a relation on the natural numbers, we can now regard every natural number occurring in the field of y as corresponding to a possible stage of the computation, where a finite number of choices is possible. For every branch of the computation $T(x)$, we have a function $\mathbb{N} \to \mathbb{N}$, that for any natural number corresponding to a stage of the computation belonging to that branch, assigns a number corresponding to the choice that has to be made at that branch. This function can be represented by a real number.

Now consider the following Hamkins-Kidder machine T' that, given a certain input, interprets it as a combination of three reals $(x \oplus y) \oplus z$. This machine will consider x as a code for an ordinal, and y as function f from $\mathbb{N}$ to $\mathbb{N}$ determining choices to make at points during this computation. This machine will simulate the nondeterministic machine T making the choices prescribed by y. At any stage of the computation, we will look for the lowest element of the current ordinal coded by x. If there is no such element, we will find this in ω steps and output 0. If there is such an element, we remember the natural number n coding it, and look through y for the value $f(n)$. The latter can be done in finitely many steps. After this, we can simulate the computation of T on the input z making the choice corresponding to the value $f(n)$ in finitely many steps and, if we find out that the computation of T would terminate with a certain value, terminate with the same value. So, for every step of the original computation, the simulated computation will take at most $\omega \cdot 2$ steps. Because the computation of T on every branch of the computation will finish before time $f(x)$, it follows that, on all inputs, the simulated computation will finish before time $\omega \cdot 2 \cdot f(x) = f(x)$.

Now, it still needs to be shown that T' actually accepts the set A nondeterministically. First, consider the situation where $a \in A$. In that case, there is a branch b of the computation by T that witnesses the fact that $a \in A$. Moreover, we know that this branch maximally has the length $f(a)$ and, because $f(a)$ is a countable ordinal, we know that there is some real x coding $f(a)$. Another real, y, will code exactly those choices made in the branch b. But now it follows that the computation of T' on input $(x \oplus y) \oplus a$ will simulate the branch in question, and thus output 1. Conversely, consider the situation where $a \notin A$. In this case, we still have the fact that all branches b of the computation by T starting from b finish within time $f(a)$ and, furthermore, that in all these branches, the output of T will be 0. There are two possibilities for a computation of T' on any input $(x \oplus y) \oplus a$: either x codes some ordinal, or it does not. If it does not, the computation can still start on a well-founded initial part of x. If this initial part is long enough, that will result in the computation of T on one the branches being simulated to the end, resulting in the output 0. If it is not long enough, the computation will at one point result in a stage where no least element can be found on x, and here, again, 0 will be output. In the situation where x codes some ordinal, either this ordinal will be long enough, resulting in one of the branches being simulated and 0 being output; or it will be too short, again resulting in the output 0. So, on any input $(x \oplus y) \oplus a$, 0 will be output.

It follows that a computation of T on a outputs 1 if and only if there is a real x such that a computation of T' on $x \oplus a$ outputs 1, in the required time, completing the theorem. □

7.1.2 Time complexity: results

Now we have shown that the two different definitions of the **NP**-classes are (mostly) equivalent, we first can mention a number of earlier results that were proven using the 'old' definition of the **NP**-classes, that is, our $\mathbf{NP}'$-classes. Note that Proposition 7.6 alone is already enough to show this: if we know that $\mathbf{NP}'_f$ contains elements that $\mathbf{P}_f$ does not contain, these elements must, due to Proposition 7.6, also be in $\mathbf{NP}_f$, so the inequality of $\mathbf{P}_f$ and $\mathbf{NP}_f$ follows directly.

Lemma 7.8. $\mathbf{NP}'_{\omega+2}\backslash\mathbf{P}_{\omega_1} \neq \emptyset$, *and hence* $\mathbf{NP}_{\omega+2}\backslash\mathbf{P}_{\omega_1} \neq \emptyset$, *and hence* $\mathbf{P}_\alpha \neq \mathbf{NP}_\alpha$ *for any* α *such that* $\alpha \geq \omega + 2$ *and* $\alpha < \omega_1$.

Proof. See [Sc, Theorem 2.10]. □

Corollary 7.9. $\mathbf{P} \neq \mathbf{NP}$.

Lemma 7.10. $\mathbf{NP}'_+\backslash\mathbf{P}_+ \neq \emptyset$, *and hence* $\mathbf{NP}_+\backslash\mathbf{P}_+ \neq \emptyset$, *and hence* $\mathbf{P}_+ \neq \mathbf{NP}_+$.

Proof. See [Sc, Corollary 2.14]. □

Lemma 7.11. $\mathbf{NP}'_{++}\backslash\mathbf{P}_{++} \neq \emptyset$, *and hence* $\mathbf{NP}_{++}\backslash\mathbf{P}_{++} \neq \emptyset$, *and hence* $\mathbf{P}_{++} \neq \mathbf{NP}_{++}$.

Proof. This is a weakening of [HaWe, Theorem 1.8]. □

7.1.3 Space complexity: definitions

Turning to the case of space complexity classes, we can again define the classes **NPSPACE** analogously to the **NP** and **PSPACE** cases. In this case we will follow [Lö] and only stick to the option of defining the space complexity classes in terms of nondeterministic machines.

Here we need an adaptation of Definition 4.18 that takes into account the existence of different branches:

Definition 7.12. If T is a nondeterministic infinite time Turing machine, and p is a halting computation by T from x, we define:

1. $\ell_\gamma(p) := \min\{\eta : \forall i \in \{1,2,3\} : c^p_{i,\gamma} \in \mathbf{L}_\eta[x]\}$
2. $\mathbf{space}(x, T, p) := \sup\{\ell_\gamma(p) : \gamma < \xi\}$

Definition 7.13. For any nondeterministic infinite time Turing machine T, we say that T is a **nondeterministic space f machine** if T uniformly halts on all inputs $x \in \mathbb{R}$, and, for any halting computation p starting from any $x \in \mathbb{R}$, we have $\mathbf{space}(x, T, p) \leq f(x)$. For any ordinal ξ, we say that T is a **nondeterministic space ξ machine** if T is a nondeterministic space f machine for the constant function f such that $f(x) = \xi$ for all x.

Definition 7.14. For any function f, we let $\mathbf{NPSPACE}_f$ denote the class of all sets of reals that are decidable by a nondeterministic space f machine. For any ordinal ξ, we let $\mathbf{NPSPACE}_\xi$ denote the class of all sets of reals that are decidable by a nondeterministic space η machine for some $\eta < \xi$.

Again, the classes **NPSPACE**, $\mathbf{NPSPACE}_+$, and $\mathbf{NPSPACE}_{++}$ are also defined analogously to the classes **PSPACE**, $\mathbf{PSPACE}_+$, and $\mathbf{PSPACE}_{++}$.

A direct result of the definitions is the following fact:

Proposition 7.15. *For all f and all α,* $\mathbf{PSPACE}^{\mathrm{HK}}_f \subseteq \mathbf{NPSPACE}^{\mathrm{HK}}_f$ *and* $\mathbf{PSPACE}^{\mathrm{HK}}_\alpha \subseteq \mathbf{NPSPACE}^{\mathrm{HK}}_\alpha$ *respectively.*

Proof. Given an infinite time Turing machine T, it is immediate that T also fulfils the definition of a nondeterministic infinite time Turing machine, in which no branching occurs. The space complexity of any halting computation of T, when seen as nondeterministic machine, thus is identical to the space complexity of T when seen as regular machine. □

Again, we can define nondeterministic ITTM_1s, nondeterministic machines with varying numbers of tapes, with additional scratch cells, with stay options, et cetera. Moreover, all of the results that were given in chapter 2 about the equivalence between different types of infinite time Turing machines (ITTM_0 and ITTM_1, both with variations in the number of tapes and the number of scratch cells) generalize to the case of nondeterministic machines. All of the stretching, contracting operations, and operations doing nothing else than simply moving left, can be directly taken over, while the actual nondeterminism is incorporated in the operation performing the actual simulation. Without a proof, we will now state that at least:

Proposition 7.16. *For any set A, A is decidable by a nondeterministic* ITTM_0 *if and only if it is decidable by a nondeterministic* ITTM_1*.*

7.2 The relationship between the classes $\mathbf{NP}_f$ and $\mathbf{NPSPACE}_f$

With all preliminary work done, we now can go on to the real problem, that is, to investigate the relationship between the classes $\mathbf{NP}_f$ and $\mathbf{NPSPACE}_f$.

The main problem here, and the reason why we have difficulties even establishing results that are trivial for regular, finite, Turing machines, is that there are machines for which we know that we have $\mathbf{time}(x, T, p) \leq \alpha$ for all halting computations p, which nonetheless can write down very complicated reals on the tape. An example of such a machine would be the machine, that, in ω steps, nondeterministically writes an 1 or a 0 on each of the ω cells of the tape. After only ω steps, every real already occurs on one of the branches. So, whereas in the earlier cases the maxim 'an algorithm never takes more space than it takes time' always held, we have no certainty about that in this case, and even counterexamples. As a result, to even prove something such as $\mathbf{NP}^{\mathrm{HK}} \subseteq \mathbf{NPSPACE}^{\mathrm{HK}}$ we have to find, for every set A in $\mathbf{NP}^{\mathrm{HK}}$, *another* machine, which also computes A, but which is guaranteed to be limited in space by $\mathbf{NPSPACE}^{\mathrm{HK}}$. Whether this is at all possible, remains an open question.

The situation, however, is not entirely hopeless: we may still try to approach things, by showing that a complicated set, such the set of all reals coding wellordered sets, occurs at a very low level in the **NPSPACE**-hierarchy. This turns out to be a feasible method indeed, and it appears that the set of non-wellordered sets appears in $\mathbf{NPSPACE}_{\omega+1}$. We first define the set WO:

Definition 7.17. The set WO is defined as $\{x \in \mathbb{R} : x \text{ codes a wellorder}\}$.

Proposition 7.18. $\overline{\mathrm{WO}} \in \mathbf{NPSPACE}_{\omega+1}$

Proof. We here use the fact that a real x codes a wellorder if and only if it is a linear order that contains no infinitely descending chains. To see if a real x is

in $\overline{\mathrm{WO}}$, therefore, it will suffice to provide a nondeterministic Turing machine with an algorithm, that will first recognize any real that is not a linear order; and following that, if an infinitely descending chain is present, recognize it on at least one of the branches.

First of all, we can easily recognize if a real x codes a linear order R by first checking if it is total, then checking if it is antisymmetric, and then checking if it is transitive. These things can all be done by only writing finite sets or possibly $\mathbb{N}$ on the tape.

For the rest of the task, the task of recognizing an infinitely descending chain $s_1, s_2, \ldots$ with $s_i R s_{i+1}$ and $s_i \in \mathrm{Field}(R)$ for $i \in \mathbb{N}$, we should first note that there must be another infinite sequence $t_1, t_2, \ldots$ with $t_i R t_{i+1}$ and $t_i \in \mathrm{Field}(R)$ for $i \in \mathbb{N}$, such that for all t_i, the index of t_i is lower than the index of t_{i+1}. The trick that we can use to keep the space complexity 'low', is to only keep track of the last element of the (possibly infinite) sequence, deleting all the earlier elements.

Now, we first need to find an initial element for the sequence, and then move along the field of the order from left to right. For each element we find, on one branch we pick it as the starting element of the sequence, and on another branch we do not. On the branch where we do, we put a flag on one of the scratch tapes at the index of the element, and put another flag on to indicate that we have found a first element. On the branch where we do not, we simply continue the search for a starting element.

When we have found at least one element for a sequence, we go on alongside the natural numbers, and for each number, we check if the element corresponding to it in the order is lower than our current lowest element. If it is, on one new branch we put a flag on this element as the new lowest element, deleting the previous lowest element, and on another branch we ignore it and keep the current lowest element.

When we reach a limit state, there are three possibilities:

1. The 'found a first element' flag is off, and (necessarily) the 'lowest element' tape is empty. We have not found an infinite descending sequence, and output 0.

2. The 'found a first element' flag is on, and there is exactly one 1 on the 'lowest element' tape. This means that this element has been chosen as 'lowest element' somewhere, but never been erased afterwards, so no lower elements have been chosen. Again, we have not found an infinite descending sequence, and output 0.

3. The 'found a first element' flag is on, and the 'lowest element' tape is empty. This means that every element that has been chosen as lowest element at one point, has been updated by a lower element afterwards. This however means that we have found an infinite descending sequence, and output 1.

This computation halts in all branches, and only writes very simple sets—finite ones containing one element, in fact—on the scratch tapes. It follows from this that the computation is in $\mathbf{NPSPACE}_{\omega+1}$. □

We now refer to the following lemma about the analytical hierarchy (for background information on the analytical hierarchy, see e.g. [Ro]):

Lemma 7.19. *For every Σ^1_1 set A, there is a recursive function f, such that $x \in A \iff f(x) \in \overline{\mathrm{WO}}$*

Proof. Corollary 20b in section 16.4 of [Ro] shows that, for every Π^1_1 set B, there is a recursive function f, such that $x \in B \iff f(x) \in \mathrm{WO}$. Now assume that A is Σ^1_1. This implies that $\overline{A}$ is Π^1_1, and there is a recursive function f, such that $x \in \overline{A} \iff f(x) \in \mathrm{WO}$. This same function f witnesses that $x \in A \iff f(x) \in \overline{\mathrm{WO}}$. □

Because all Σ^1_1 sets are reducible to the compliment of WO by a recursive function, we can generalize Lemma 7.19 to the following result:

Proposition 7.20. *For any Σ^1_1 set A, we have $A \in \mathbf{NPSPACE}_{\omega+2}$.*

Proof. Because A is Σ^1_1, we know that there must be some recursive function f, such that $x \in A \leftrightarrow f(x) \in \overline{\mathrm{WO}}$. We can now proceed by first computing the function f on the input, which takes us ω time, and then performing the earlier algorithm to search an infinite descending sequence. Because, during the latter part of the computation, the content of the tape will only ever differ finitely many cells from the earlier result $f(x)$, we are guaranteed that the space complexity of any of the snapshots will never go beyond the $\omega + 2$ level. It now follows that, indeed, $A \in \mathbf{NPSPACE}_{\omega+2}$. □

Which gives us the following result:

Proposition 7.21. *We have $\mathbf{NP}_+ \subseteq \mathbf{NPSPACE}_{\omega+2}$ and hence, for any ordinal α such that $\omega + 1 \leq \alpha \leq \omega_1^{\mathrm{CK}}$, also $\mathbf{NP}_\alpha \subseteq \mathbf{NPSPACE}_\alpha$.*

Proof. Theorem 5 from [DeHaSc] shows that $\mathbf{NP'}_+ = \Sigma^1_1$, and because the function $f_0(x) = \omega_1^x$ satisfies the conditions of Propositions 7.6 and 7.7, $\mathbf{NP}_+ = \Sigma^1_1$ follows.

From this and Proposition 7.20, it is a direct consequence that $\mathbf{NP}_+ \subseteq \mathbf{NPSPACE}_{\omega+2}$, and the desired result follows. The second result then directly follows from the fact that $\mathbf{NP}_\alpha \subseteq \mathbf{NP}_+$ if $\alpha \leq \omega_1^{\mathrm{CK}}$, and the fact that $\mathbf{NPSPACE}_{\omega+1} \subseteq \mathbf{NPSPACE}_\alpha$ if $\alpha \geq \omega + 1$. □

Of course, this is not much: we have shown one instance of the problem $\mathbf{NP}_\alpha \subseteq \mathbf{NPSPACE}_\alpha$, a problem which, in the case of finite Turing machine computations and, as we will soon see, also in the case of nondeterministic Koepke machines, is trivially easy to solve.

7.3 Nondeterministic computation in the case of Koepke machines

In the case of Koepke machines, things seem to be looking a little bit brighter. Here we have a definition of space complexity that is simpler and more intuitive, and it turns out that we have a few analogues for some of the theorems about deterministic Koepke machine computation, such as Proposition 6.16 and Proposition 6.17. We start out by giving the—unsurprising—definitions, and continue by presenting a number of results.

7.3.1 Definitions

Definition 7.22. A **nondeterministic Koepke machine** is defined like a regular Koepke machine, but here again δ is a relation between $Q \times \{0,1\}^3$ and $Q \times \{0,1\}^3 \times \{L, R\}$, rather than a function.

We again define the notions of **partial computations**, **halting computations**, **uniform halting**, and **uniformly halting in time** f, like we did in the case of nondeterministic Hamkins-Kidder machines.

Definition 7.23. For any nondeterministic Koepke machine T, we say that T is a **nondeterministic time** f **Koepke machine** if for all x, the computation of T on x uniformly halts at time $f(x)$. For any ordinal ξ, we say that T is a **nondeterministic time** ξ **Koepke machine** if T is a nondeterministic time f Koepke machine for the constant function f with $f(x) = \xi$ for all x.

Definition 7.24. For any function f, we let $\mathbf{NP}^{\mathrm{K}}_f$ denote the class of all sets of reals that are decidable by a nondeterministic time f Koepke machine. For any ordinal ξ, we let $\mathbf{NP}^{\mathrm{K}}_\xi$ denote the class of all sets of reals that are decidable by a nondeterministic time η Koepke machine for some $\eta < \xi$.

Definition 7.25. For any nondeterministic Koepke machine T, we say that T is a **nondeterministic space** f **Koepke machine** if, for all branches of the computation, and for all x, the position of the head always stays at or below $f(x)$. For any ordinal ξ, we say that T is a **nondeterministic space** ξ **Koepke machine** if T is a nondeterministic space f Koepke machine for the constant function f with $f(x) = \xi$ for all x

Definition 7.26. For any function f, we let $\mathbf{NPSPACE}^{\mathrm{K}}_f$ denote the class of all sets of reals that are decidable by a nondeterministic space f Koepke machine. For any ordinal ξ, we let $\mathbf{NPPACE}^{\mathrm{K}}_\xi$ denote the class of all sets of reals that are decidable by a nondeterministic time η Koepke machine for some $\eta < \xi$.

For Koepke machines, at least it becomes evident directly that the 'sanity check' 'time complexity classes are smaller than or equal to the corresponding space complexity classes' holds. Here, just like in the case of finite Turing machines, but unlike in that of Hamkins-Kidder machines, one 'time unit' extra in a computation can mean at most one 'space unit' extra:

Proposition 7.27. *For all f and all α, we have $\mathbf{NP}^{\mathrm{K}}_f \subseteq \mathbf{NPSPACE}^{\mathrm{K}}_f$ and $\mathbf{NP}^{\mathrm{K}}_\alpha \subseteq \mathbf{NPSPACE}^{\mathrm{K}}_\alpha$ respectively.*

Proof. It is easy to see by ordinal induction on α, we have $h^T_\alpha(x) < \alpha$ for any x and any α on all branches. From this, it follows immediately from the definitions that $\mathbf{space}(x, T, p) < \mathbf{time}(x, T, p)$ for all halting computations p. Hence, for any ordinal α, we have $A \in \mathbf{NPSPACE}^{\mathrm{K}}_\alpha$ if $A \in \mathbf{NP}^{\mathrm{K}}_\alpha$, and for any function f, we have $A \in \mathbf{NPSPACE}^{\mathrm{K}}_f$ if $A \in \mathbf{NP}^{\mathrm{K}}_f$. □

Also, Proposition 7.15 carries over directly to the case of Koepke machines.

Proposition 7.28. *For all f and all α, $\mathbf{PSPACE}^{\mathrm{HK}}_f \subseteq \mathbf{NPSPACE}^{\mathrm{HK}}_f$ and $\mathbf{PSPACE}^{\mathrm{HK}}_\alpha \subseteq \mathbf{NPSPACE}^{\mathrm{HK}}_\alpha$ respectively.*

Proof. Like in Proposition 7.15. □

7.3.2 Results

To start, we notice that any set that is nondeterministically decidable by a Hamkins-Kidder machine, is decidable by a nondeterministic Koepke machine using only space $\omega + 2$:

Proposition 7.29. $\mathbf{NDec}^{\mathrm{HK}} \subseteq \mathbf{NPSPACE}^{\mathrm{K}}_{\omega+2}$

Proof. Assume that $A \in \mathbf{NDec}^{\mathrm{HK}}$. By Proposition 7.16, we may assume that A is decided by a nondeterministic ITTM_1 machine that is able to recognize when it is at a limit stage. We now can simulate A on a nondeterministic Koepke machine with tape length $\omega + 1$ by moving left ω times at any limit stage, and besides this, simply computing the original program used by the Hamkins-Kidder machine. □

However, even stronger than in the previous case, where the longer tapes did not give us any extra power until length ζ^x, it turns out that now, we do not gain any strength from the longer tapes even until length ω_1.

Proposition 7.30. $\mathbf{NPSPACE}^{\mathrm{K}}_{\omega_1} \subseteq \mathbf{NDec}^{\mathrm{HK}}$

Proof. Assume that we have $A \in \mathbf{NPSPACE}^{\mathrm{K}}_{\omega_1}$. In that case, there is a countable ordinal α, such that all branches of the machine T, which decides A, take less than α space on any input $x \in \mathbb{R}$.

We can now run a nondeterministic Hamkins-Kidder program, that first nondeterministically writes a real out in ω steps, such that, after ω steps, every real occurs on one branch of the tape. After this, on every branch we start checking if the real we have codes an ordinal: if it does not, output 0. If it does, we can simulate the computation of the machine T using this ordinal as our tape: if the program ever finishes without running out of tape, give the obtained output. If the program does not finish without running out of tape, we can output 0.

Because A is computable within countable space by a Koepke machine, we know that, for every x, on at least one of the branches, the space in the simulated machine will be enough. As a result, if $x \in A$, there will be at least one such branch where this will be recognized and the Hamkins-Kidder machine will actually output 1 as a result. □

A result of this is that, in the $\mathbf{NPSPACE}^{\mathrm{K}}$ hierarchy, we will have equality all the way between $\omega + 2$ and ω_1:

Proposition 7.31. *For any ordinal α such that $\omega + 2 < \alpha \leq \omega_1$, we have that* $\mathbf{NPSPACE}^{\mathrm{K}}_{\alpha} = \mathbf{NDec}^{\mathrm{HK}}$.

Proof. This follows directly from the Propositions 7.29 and 7.30. □

And, because we know that $\mathbf{NDec}^{\mathrm{HK}}$ is a larger set than $\mathbf{Dec}^{\mathrm{HK}}$, we also obtain the following result relating $\mathbf{NPSPACE}^{\mathrm{K}}$ and $\mathbf{PSPACE}^{\mathrm{K}}$:

Proposition 7.32. *For any ordinal α such that $\omega + 2 < \alpha < \Sigma$, we have that* $\mathbf{PSPACE}^{\mathrm{K}}_{\alpha} \subsetneq \mathbf{NPSPACE}^{\mathrm{K}}_{\alpha}$.

Proof. On one side, we know that $\mathbf{PSPACE}^{\mathrm{K}}_{\alpha} \subseteq \mathbf{Dec}^{\mathrm{HK}}$, and hence that $h \neq \mathbf{PSPACE}^{\mathrm{K}}_{\alpha}$. On the other side, we know that $h \in \mathbf{NDec}^{\mathrm{HK}}$, and hence that $h \in \mathbf{NPSPACE}^{\mathrm{K}}_{\alpha}$. □

Furthermore, we obtain the following analogue to Proposition 6.13:

Proposition 7.33. $\mathbf{NP}^{\mathrm{HK}}_{\lambda} = \mathbf{NP}^{\mathrm{K}}_{\lambda}$.

Proof. Assume that A is decidable by a nondeterministic time η Koepke machine, where $\eta < \lambda$ or, in other words, η is a writable ordinal. Clearly this computation can only use at most η cells on any of the branches. Say, η is writable by a Hamkins-Kidder machine in time θ. Now θ too must be a writable ordinal, due to Corollary 3.20, and the fact that it is clockable as there is a computation starting on input 0 finishing in time θ. After writing a code for η on the tape in θ steps, we can continue by simulating the Koepke algorithm by simulating a tape of size η, in time less than η. Thus, we can decide A with a nondeterministic Hamkins-Kidder machine in time $\theta + \eta$, which is again a writable ordinal by Proposition 3.22, and hence smaller than λ. □

We also have:

Proposition 7.34. *For any ordinal $\alpha < \lambda$, we have* $\mathbf{NP}^{\mathrm{HK}}_{\alpha} \subsetneq \mathbf{NP}^{\mathrm{HK}}_{\lambda}$.

Proof. Because $\alpha < \lambda$, there is a clockable ordinal $\eta \geq \alpha$, and by [HaLe, Many Gaps Theorem 3.6], there are at least η gaps in the clockable ordinals of size η. So there must be a gap in the clockable ordinals starting at some ordinal larger than η. Let β be the least ordinal larger than η starting a gap in the clockable ordinals of size larger than η: β again is a writable ordinal, and hence smaller than λ. From [DeHaSc, Theorem 17], it follows that $\mathbf{P}^{\mathrm{HK}}_{\beta} = \mathbf{NP}^{\mathrm{HK}}_{\beta} \cap \mathbf{coNP}^{\mathrm{HK}}_{\beta}$, and from [DeHaSc, Corollary 18] it follows that $\mathbf{P}^{\mathrm{HK}}_{\lambda} = \mathbf{NP}^{\mathrm{HK}}_{\lambda} \cap \mathbf{coNP}^{\mathrm{HK}}_{\lambda}$. But, because $\mathbf{P}^{\mathrm{HK}}_{\beta} \subsetneq \mathbf{P}^{\mathrm{HK}}_{\lambda}$, it follows that $\mathbf{NP}^{\mathrm{HK}}_{\beta} \cap \mathbf{coNP}^{\mathrm{HK}}_{\beta} \subsetneq \mathbf{NP}^{\mathrm{HK}}_{\lambda} \cap \mathbf{coNP}^{\mathrm{HK}}_{\lambda}$, and hence that both $\mathbf{NP}^{\mathrm{HK}}_{\beta} \subsetneq \mathbf{NP}^{\mathrm{HK}}_{\lambda}$ and $\mathbf{coNP}^{\mathrm{HK}}_{\beta} \subsetneq \mathbf{coNP}^{\mathrm{HK}}_{\lambda}$, from which the initial claim follows. □

Which gives us the following result relating $\mathbf{NP}^{\mathrm{K}}$ and $\mathbf{NPSPACE}^{\mathrm{K}}$:

Proposition 7.35. *For any ordinal α such that $\omega + 2 < \alpha < \lambda$, we have that* $\mathbf{NP}^{\mathrm{K}}_{\alpha} \subsetneq \mathbf{NPSPACE}^{\mathrm{K}}_{\alpha}$.

Proof. On one side, we have that $\mathbf{NP}^{\mathrm{K}}_{\alpha} \subsetneq \mathbf{NP}^{\mathrm{K}}_{\lambda}$, and hence $\mathbf{NP}^{\mathrm{K}}_{\alpha} \subsetneq \mathbf{NDec}^{\mathrm{HK}}$. On the other hand, we have that $\mathbf{NDec}^{\mathrm{HK}} = \mathbf{NPSPACE}^{\mathrm{K}}_{\alpha}$, from which the result follows. □

Chapter 8

Conclusions

At the beginning of this thesis, we set out to look at the relationship between the classes of the family **P** and those of the family **PSPACE**. We found that, in the case of Hamkins-Kidder machines and the case of Koepke machines, the inclusions $\mathbf{P}_f \subseteq \mathbf{PSPACE}_f$ generally held. In the case of Hamkins-Kidder machines, the only exceptions are finite ordinals α functions f where the range of f contains finite ordinals. In the case of Koepke machines, both inclusions hold, just like in the case of ordinary, finite, Turing machines, trivially, as a result of the general fact that 'a computation cannot use more space than time'.

For a number of cases, we also proved the strict inclusions $\mathbf{P}_f \subsetneq \mathbf{PSPACE}_f$ and $\mathbf{P}_\alpha \subsetneq \mathbf{PSPACE}_\alpha$. In the cases of Hamkins-Kidder machines, these cases are: (a) ordinals in between $\omega + 2$ and $\omega + 2$, (b) recursive successor ordinals larger than $\omega + 1$, and (c) successor ordinals α such that there is a clockable ordinal β such that $\alpha \leq \beta + 3$ and α is smaller than the next admissible ordinal after β. In the case of Koepke machine, the strict inclusion is proven to hold for all ordinals α such that $\omega + 1 \leq \alpha \leq \lambda$.

For other functions f, we have proved that equality holds between $\mathbf{P}_f$ and $\mathbf{PSPACE}_f$. In the case of Hamkins-Kidder machines, this was shown for all f such that $f(x) \leq \lambda^x$ for all x, and in the case of Koepke machines, this was shown for all f such that $\lambda^x \leq f(x) \leq \Sigma^x$.

Besides the relationship between the classes **P** and **PSPACE**, we also looked at the relationship between the classes **NP** and **NPSPACE**, and the relationship between the classes **PSPACE** and **NPSPACE**. Regarding the relationship between **NP** and **NPSPACE**, the results for Hamkins-Kidder machines are few and far between: so far, the only thing that has been shown, was the inclusion $\mathbf{NP}_\alpha \subsetneq \mathbf{NPSPACE}_\alpha$ for a very limited range of ordinals. In the case of Koepke machines, however, the inclusions $\mathbf{NP}_\alpha \subseteq \mathbf{NPSPACE}_\alpha$ and $\mathbf{NP}_f \subseteq \mathbf{NPSPACE}_f$ again hold trivially, and furthermore, for ordinals α with $\omega + 2 < \alpha < \lambda$, we proved that the strict inclusion $\mathbf{NP}_\alpha \subsetneq \mathbf{NPSPACE}_\alpha$ holds.

Regarding the relationship between **PSPACE** and **NPSPACE**, the inclusions $\mathbf{PSPACE}_f \subseteq \mathbf{NPSPACE}_f$ and $\mathbf{PSPACE}_\alpha \subseteq \mathbf{NPSPACE}_\alpha$ hold trivially for all α and all f, both in the case of Hamkins-Kidder and Koepke machines, as a direct result of the fact that any Hamkins-Kidder machine and any Koepke machine also fulfils the requirements of a nondeterministic Hamkins-Kidder machine and a nondeterministic Koepke machine, respectively. In the case of Koepke machines, moreover, we showed that for all ordinals α with

$\omega + 2 < \alpha < \Sigma$, the strict inclusion $\mathbf{PSPACE}_\alpha \subsetneq \mathbf{NPSPACE}_\alpha$ holds.

Other than establishing the relationship between different complexity classes, a large part of this thesis also was—partly because these results were needed to give sufficiently formal proofs of the later theorems—concerned with proving the equality of variants in the models of Infinite Turing computation. An important equivalence shown is that between the ITTM_0 and ITTM_1 models, and variants of those models with different numbers of tapes and/or scratch cells.

Still, quite a few questions so far remain unsolved: so far, we have not managed to find the truth or falsity of either $\mathbf{P} \subsetneq \mathbf{PSPACE}$, $\mathbf{P}_+ \subsetneq \mathbf{PSPACE}_+$, or $\mathbf{P}_{++} \subsetneq \mathbf{PSPACE}_{++}$. The main technique we used to prove the equality of $\mathbf{P}_\alpha$ and $\mathbf{PSPACE}_\alpha$, namely showing that a certain halting set is in $\mathbf{PSPACE}_\alpha$ but not in $\mathbf{P}_\alpha$, does not appear to work here.

Furthermore, we have not looked at all at the computation of Koepke machines beyond stage ω_1. Especially in the case of Koepke machines, though, many of the 'main' questions were solved for important classes of countable ordinals.

Other than this, there seems to be a large amount of theorems that appear to fit in the 'easy but tedious' category—many of which have not been proven rigorously yet. For example, here we can think of the invariance of clockable, accidentally writable, and eventually writable ordinals between the different type of models: especially the latter seem to be easily proven with the right type of simulation.

All these open and unsolved issues could, in principle, be material for further investigation.

Appendix A

Variations in the definitions of $\mathbf{P}_f$ and $\mathbf{PSPACE}_f$

A.1 Variations in the definition of $\mathbf{P}_f$

In chapter 4, we defined $\mathbf{P}_f$ as the family of all sets of reals A, such that A is decidable by a time f machine, that is, a machine T, such that $\mathbf{time}(x, T) \leq f(x)$ for all x.

In [Sc] however, we find another definition, namely:

> We say that $A \in \mathbf{P}_f$ if there is a Turing machine T such that (a) $x \in A$ if and only if T accepts x and (b) T halts on all inputs x after $< f(x)$ many steps.

In this paper, it is not made clear what exactly is meant with 'after α many steps'; however, in section 3 of [HaLe], in the definition of clockable ordinals, 'halting in exactly α steps' is clarified as 'the αth step of computation is the act of changing to the halt state'. Because 'the αth step of computation' must be equal to 'the step going from stage α to stage $\alpha + 1$'[1], this would, in our terminology be equivalent to $q^T_{\alpha+1} = q_{\mathrm{f}}$.

So the definition in [Sc] is, in our terminology, equivalent to: '$A \in \mathbf{P}^f$ if and only if there is an infinite time Turing machine T, such that T accepts A, and, for all x, $\mathbf{time}(x, T) < f(x) + 1$'—as 'halting within α steps' only implies the halt state must be reached at stage $\alpha + 1$. However, $\mathbf{time}(x, T) < f(x) + 1$ can be written as $\mathbf{time}(x, T) \leq f(x)$, so this definition is equivalent to the definition in this thesis.

In [HaWe], we find yet another definition:

> We say that $A \in \mathbf{P}_f$ if there is an (infinite time) Turing machine computable function ϕ_e so that (i) A is decidable by ϕ_e, that is $x \in A$ if and only if $\phi_e(x) \downarrow 1$, and (ii) for all $z \in {}^{\omega}2$, $\phi_e(z) \downarrow$ in at most $f(z)$ many steps.

Here the 'less than' clause is replaced by an 'at most' clause: in our terminology, thus, this definition can be rephrased as: '$A \in \mathbf{P}^f$ if and only if there

[1] Otherwise, no limit ordinal could possibly be clockable as there would be no αth stage for limit ordinals

is an infinite time Turing machine T, such that T accepts A, and, for all x, $\mathbf{time}(x, T) \leq f(x) + 1$'.

A third definition can be found in [Lö]: here, like in this thesis, a set is considered to be in $\mathbf{P}_f$ if and only if it is decidable by a time f machine. However, the definition of a time f machine in [Lö] differs from our definition: there, a time f machine is defined as a machine T such that $\mathbf{time}(x, T) < f(x)$ for all x.

Because of this, for successor ordinals β and for functions f, it follows that the classes P_f and P_α from the definition in [HaWe] correspond to the classes P_{f+1} and $\mathrm{P}_{\alpha+1}$ according to the definition in [Sc], and to the classes P_{f+2} and $\mathrm{P}_{\alpha+2}$ according to the definition in [Lö].

The classes $\mathbf{P}_\alpha$ for limit ordinals α, on the other hand, do not differ between the various definitions, as $\mathbf{P}_\alpha$ is defined as 'decidable by a time ξ machine for some $\xi < \alpha$', and if $\xi < \alpha$, then $\xi + 1 < \alpha$ and $\xi + 2 < \alpha$ as α is a limit ordinal.

Likewise, in the definition of the weak halting problem h_α relativized to α, it might appear to be a deviation from the earlier definition that we made the requirement $\mathbf{time}(e, e) \leq \alpha$, and not $\mathbf{time}(e, e) < \alpha$. However, $\mathbf{time}(e, e) \leq \alpha$ corresponds directly to the notion 'halts in less than α many steps' that was used in [DeHaSc].

A.2 Variations in the definition of $\mathbf{PSPACE}_f$

Also with regard to the definition of the space complexity classes, the definitions in this thesis differ from those in [Lö], and they do so in exactly the same way our time complexity classes differ from those in [Lö]. The operation $\mathbf{space}(x, T)$ is still defined identically, but whereas we define a space f machine as a machine T such that $\mathbf{space}(x, T) \leq f(x)$, in [Lö] this is defined as $\mathbf{space}(x, T) < f(x)$[2].

If we use the definitions from [Lö] in both the cases of time and space complexity, almost all of the results in this thesis will still hold. There are a few theorems where this is not directly evident, or where small changes have to be made:

- Proposition 5.13 has to be restated as 'For every ordinal α, we have $h_\alpha \notin \mathbf{P}_{\alpha+2}$'.
- Corollary 5.7, still holds, as we still have $\mathbf{space}(x, T) \leq \mathbf{time}(x, T)$.
- In Theorems 5.11 and 5.12, all occurrences of $\mathbf{PSPACE}_{\omega+2}$ have to be replaced by $\mathbf{PSPACE}_{\omega+3}$.
- In Proposition 5.13, we are only guaranteed that $h_\alpha \in \mathbf{PSPACE}_{\alpha+2}$, and hence, in Theorem 5.14, we in first instance can only obtain $\mathbf{P}_\alpha \subsetneq \mathbf{PSPACE}_\alpha$ for successors of successor ordinals larger than $\omega+2$. Likewise, Theorem 5.16 only works for successors of successor ordinals between $\beta+4$ and the next admissible after β.
- In Proposition 7.18, we have to replace $\omega+1$ with $\omega+2$, and in Proposition 7.20, we have to replace $\omega + 2$ with $\omega + 3$.

[2] Actually, no precise definition is given at all in [Lö], but this definition is implied as the notation with $<$ was given in the definition of $\mathbf{NP}_f$, which *was* given explicitly.

If, on the other hand, we use the definition of space complexity classes from [Lö] in combination with the definition of time complexity classes from [Sc], things turn out to be more problematic: for example, in this case, we are not guaranteed that $\mathbf{P}_f \subseteq \mathbf{PSPACE}_f$, but only $\mathbf{P}_f \subseteq \mathbf{PSPACE}_{f+1}$.

Bibliography

[Co] S. Barry Cooper, *Computability Theory*, Chapman & Hall/CRC, 2004

[DeHaSc] Vinay Deolalikar, Joel David Hamkins, Ralf-Dieter Schindler, $\mathbf{P} \neq \mathbf{NP} \cap \text{co}-\mathbf{NP}$ *for infinite time Turing machines*, Journal of Logic and Computation 15 (2005), pp. 577-592

[HaLe] Joel David Hamkins and Andy Lewis: *infinite time Turing machines*, Journal Of Symbolic Logic 65 (2000), pp. 567-604

[HaSc] Joel David Hamkins and Daniel Evan Seabold: *Infinite Time Turing Machines With Only One Tape*, Mathematical Logic Quarterly 47 (2001), pp. 271-287

[HaSe] Joel David Hamkins, Daniel Evan Seabold, *infinite time Turing machines With Only One Tape*, *Mathematical Logic Quarterly*, 47 (2001) 2, pp. 271-287

[HaWe] Joel David Hamkins and Philip D. Welch, $\mathbf{P}_f \neq \mathbf{NP}_f$ *for almost all* f, Mathematical Logic Quarterly, 49(2003) 5, pp. 536-540

[Ko] Peter Koepke, *Turing computations on ordinals*, Bulletin of Symbolic Logic 11 (2005), pp. 377-397

[Ku] Kenneth Kunen, *Set Theory, An Introduction to Independence Proofs*, North-Holland, 1980

[Lö] Benedikt Löwe: *Space bounds for infinitary computation*, in *Logical Approaches to Computational Barriers*, Springer (2006), pp. 319-329

[Ro] Hartley Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill Book Company, 1967

[Sc] Ralf Schindler, $\mathbf{P} \neq \mathbf{NP}$ *for infinite time Turing machines*, Monatshefte der Mathematik 139 (2003), pp. 335-340

[We] Philip D. Welch, *The Length of infinite time Turing machine Computations*, Bulletin of the London Mathematical Society 32 (2000), pp. 129-136

[We2] Philip D. Welch, *Bounding lemmata for non-deterministic halting times of transfinite Turing machines*