Three models of ordinal computability

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von
Benjamin Seyfferth
aus
Nürnberg

Bonn, Dezember 2012

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen
Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Peter Koepke
2. Gutachter: Prof. Dr. Stefan Geschke

Tag der Promotion: 12. April 2013

Erscheinungsjahr: 2013

Abstract

In this thesis we expand the scope of ordinal computability, i.e., the study of models of computation that are generalized to infinite domains. The discipline sets itself apart from classical work on generalized recursion theory by focusing strongly on the computational paradigm and an analysis in elementary computational steps. In the present work, two models of classical computability of which no previous generalizations to ordinals are known to the author are lifted to the ordinal domain, namely λ-calculus and Blum-Shub-Smale machines. One of the multiple generalizations of a third model relevant to this thesis, the Turing machine, is employed to further study classical descriptive set theory. The main results are:

An ordinal λ-calculus is defined and confluency properties in the form of a weak Church-Rosser theorem are established. The calculus is proved to be strongly related to the constructible hierarchy of sets, a feature typical for an entire subfamily of models of ordinal computation.

Ordinal Turing machines with input restricted to subsets of ω are shown to compute the Δ^1_2 sets of reals. Conversely, the machines can be employed to reprove the absoluteness of Σ^1_2 sets (Shoenfield absoluteness) and basic properties of Σ^1_2 sets. New tree representations and new pointclasses defined by the means of ordinal Turing computations are introduced and studied.

The Blum-Shub-Smale model for computation on the real numbers is lifted to transfinite running times. The supremum of possible runtimes is determined and an upper bound on the computational strength is given.

Summarizing, this thesis both expands the field of ordinal computability by enlarging its palette of computational models and also connects the field further by tying in the new models into the existing framework. Questions that have been raised in the community, e.g. on the possibility of generalizations of λ-calculus and Blum-Shub-Smale machines, are addressed and answered.

Contents

Chapter 1

Introduction

1.1 Infinity and computation

Computational reasoning in the traditional sense relies heavily on the requirement of finiteness. A *finite* program on *finite* data needs to run for only *finitely* many steps before halting. Relaxing these constraints leads to what we call models of *hypercomputation.* For instance, the classical oracle machines do away with the first finiteness restriction while retaining the latter two; a possibly infinite set of integers is supplied together with the program and during the course of the computation the program may ask whether some pieces of data in the memory belong to that set or not.

The notion of *ordinal numbers*, or *ordinals*, is a mathematical concept of infinity that bears great similarities to natural numbers. It captures the idea of 'counting' along infinite sets. While the juxtaposition of the words 'counting' and 'infinite' appears unintuitive at first, objects that lend themselves to counting along their elements appear naturally in mathematics. A *linear order* is a set which can be imagined as points in a line. If the line is imagined to run from left to right, out of any two elements we can call the one to the left 'smaller' than the other. Examples include, of course, the common numbers, be it rational, real, integer, or natural. Now the reals, for example, do not lend themselves well to counting; even if we fix a point where we would like to start counting (and, in essence, forget all numbers left of that point), there is no proper concept of 'next element'. We solve both problems, the lack of least element and the lack of next element, by restricting the notion of linear orders to that of *well-orders*: A well-order is a linear order where every nonempty subset, including the entire set itself, has a least element. So we can start counting at the least element and continue with the least element of the subset that arises by leaving out the previously counted elements. Up to isomorphism, the set of natural numbers $\mathbb{N}$ is in fact the smallest infinite well-order. We can define larger well-orderings by gluing two copies of $\mathbb{N}$ together, declaring all numbers of the second copy greater than any number in the first:

$$0, 1, 2, 3, \ldots 0', 1', 2', 3', \ldots$$

Counting across the dots, or *limits*, requires an application of the well-foundedness property: Since it is certain that the counting process will eventually reach every natural number, one can continue with the least element that cannot be reached by iterations of '+1' on 0, in this case $0'$. A particular type of well-orders built up in this way are the ordinal numbers used in set theory. Ordinals characterize well-orders, that is, for any well-order, there is an order-isomorphic ordinal. They also build the back-bone of the typical hierarchical models of set theory: The prototypical model V of the set theory ZFC can be represented as an iteration of power set operations along all ordinals, starting with the empty set.

Set theoretically, the ordinals are defined as those sets that are transitive (i.e., all elements are in fact subsets) and whose elements are well-ordered by the $\in$-relation. They take the following form:

$$\begin{aligned} 0 &= \emptyset \\ 1 &= 0 \cup \{0\} = \{0\} \\ 2 &= 1 \cup \{1\} = \{0,1\} \\ &\vdots \\ \omega &= \mathbb{N} \\ \omega + 1 &= \omega \cup \{\omega\} \\ &\vdots \end{aligned}$$

The ordinals form a well-ordered class Ord that is equipped with a certain *ordinal arithmetic*. It is set-theoretically possible to perform *transfinite recursion and induction* along Ord. For a proper introduction, we refer the interested reader to one of the standard textbooks on set theory, for instance the comprehensive [Jec03].

Ordinal computability is a particular branch of hypercomputation that studies computations along ordinals. This, again, is not as unintuitive as it may sound. One could, for instance, imagine a computer to write out the complete decimal expansion of some computable irrational number and carrying out real arithmetic precisely (instead of approximatively) while still working in distinct, elementary computational steps dealing with each decimal place at a given time. For a model along these lines, the ordinals provide the proper domain for both time, i.e., the length of the computation, and space, i.e., the required memory of the computation. Another, more interesting application is the following: Given a pair of algorithm and input, classical computability cannot decide whether the computation halts after finitely many steps or enters an infinite loop (a famous negative answer to Hilbert's 'Entscheidungsproblem'). A computation along an infinite ordinal, however, could solve this finite halting problem simply by going through all, possibly infinitely many, computational steps to see if halting occurs. It can then continue computing with this newly-established information (as an example see the theory of ITTMs [HL00]). Thus, by allowing computation along larger well orders, the idea of step-by-step computation is retained, while greatly widening the strength of the computational paradigm.

1.2 Classical computability

Let us first briefly establish the classical models of computability relevant for this thesis.

1.2.1 The λ-calculus

Developed in the 1930s by Alonzo Church, λ-calculus was one of the first concepts able to capture the idea of what Church called an 'effectively calculable function' and to give a negative answer to Hilbert's Entscheidungsproblem [Chu36]. Apart from being of great importance in the early development of computability (in particular in the influential work of Kleene), λ-calculus made its mark in the emerging discipline of theoretical computer science: The intentional lack of distinction between functions and data in λ-calculus is the core concept in what is known today as functional programming languages (e.g., Lisp), and at least rudimentary functional programming capability can be found in almost every one of today's programming languages. Although a priori being a recursion scheme, from the modern perspective λ-calculus can rightfully be called a model of computation.

In λ-calculus, one only treats objects of one type, the so-called *λ-terms* or just terms. A term may play the role of a bit of data (say, a natural number), or a function on natural numbers, or a functional, or a functional of even higher degree, all depending on the syntactical context. Terms are built up inductively as follows: Atomic elements are variables. Given some term M that may or may not contain the variable x, we can construct a new term via *λ-abstraction*: The term $\lambda x.M$ can be interpreted as the function that 'maps x to $M(x)$'. Terms may be applied to each other - the term (P, Q) could be interpreted as value of the term P interpreted as function applied to Q. To give meaning to the interpretations hinted at, *rules of conversion* are specified: Apart from bookkeeping rules dealing with renaming of variables, the integral rule is known as *β-conversion, β-reduction,* or the *β-rule*: A term $(\lambda x.M, N)$ may be reduced to $M\frac{N}{x}$, i.e., the term resulting from M by replacing all occurrences of x by N. If the β-rule cannot be applied to some term and all of its subterms, we say that this term is *in normal form*.

Reductions play the role of computations in this context: To define functions on $\mathbb{N}$, one can declare certain terms to represent the numbers. For instance, we could set $\underline{\underline{n}} = \lambda f.\lambda x.\underbrace{(f, (f, \ldots (f}_{n\text{-times}}, x)\ldots)$. These terms are in normal form and represent the n-fold application of the first argument to the second. A *λ-definable function f on* $\mathbb{N}$ is one for which a term F exists, such that for all n, $(F, \underline{\underline{n}})$ can be reduced to $\underline{\underline{f(n)}}$. A major result due to Church and Rosser is that, if the term is reducible to a normal form, a certain pattern of β-rule applications always yields a normal form and that this form is unique.

In this thesis, we shall flesh out a generalization of these concepts to ordinal numbers. This is work taken up jointly with Tim Fischbach in 2010 and completed by the author in 2012. The author is not aware of a previous work on generalizing λ-calculus to the ordinals.

1.2.2 The Turing machine

The Turing machine was developed independently of Church's λ-calculus, again with the primary purpose of answering the Entscheidungsproblem [Tur36]. As an abstraction of a human 'computer', mechanically carrying out algorithmic calculations on an idealized piece of paper, its approach is fundamentally different from Church's. Surprisingly, the two models are identical in strength. Both authors claimed to have captured the very essence of computable or effectively calculable functions on natural numbers, and this assertion is known today as the Church-Turing thesis.

For most modern readers, however, Turing's approach is what seems more intuitive and what makes the Church-Turing thesis more plausible. This may be due to Turing's clear-cut abstraction from the schematical calculations everybody is taught in school or due to the ubiquitous electronic computers of modern day, which very strongly resemble Turing machines: A *Turing machine* consists of a one-side infinite strip of *tape* that is subdivided into single *cells*. Every cell contains a bit of data, for instance the numbers either 0 or 1. On this tape, a *read-write head* (initially resting on cell 0) moves about which has a finite number of *states* and at any time reads a single cell. A look-up table lists, for a combination of state and current cell content, an *instruction* of the following form: Write a certain symbol in the current cell, move the head by one cell to the left or right, and change into a certain state. If the program wants to change to a state for which the table does not have any entry, the computation *halts*. This table is what is called the machine's *program*. With only slight modifications, this is a workable model for the hard- and software so familiar to us.

The standard definition for computable functions goes as follows: A function on natural numbers is computable if there is a Turing program such that for every n the computation by this program with initial tape content n (e.g., coded as a 1 in the n-th cell and 0's everywhere else) halts and puts out $f(n)$, i.e., the tape is empty except for a 1 in cell number $f(n)$. Functions of higher arity can be implemented by fixing some coding of multiple numbers onto the tape, e.g., by arithmetically coding tuples into single natural numbers.

With this model we can define:

- Enumerable or semi-decidable sets of natural numbers ('enumerable reals'): A set $A \subseteq \omega$ is enumerable if there is a Turing program that halts only on the members of A.
- Decidable sets of natural numbers or decidable reals: A program P gives output 1 on all n that are in the set and 0 on all other natural numbers. Also, a set is decidable exactly if both itself and its complement is enumerable.
- Enumerable or semi-decidable sets of reals: A real, coded as a subset of ω, can be put on the tape as a infinite series of 0s and 1s representing its characteristic function. A set of reals A is enumerable if there is a program that halts exactly on the initial tape contents that correspond to elements of A.
- Decidable sets of reals: Again, we want a program with output 1 on members and 0 on non-members, inputs being characteristic functions of subsets of ω. Also here we have a characterization of decidable as both enumerable and 'co-enumerable'.

The last two notions take rather trivial forms in the classical theory but will be of great interest in the generalization to ordinals studied in this thesis.

An often used feature of Turing machines is the existence of a so-called *universal machine*. One can code Turing programs into natural numbers such that there is a Turing program U so that for any program P and input n, the computation by U on inputs n and a code for P halts if and only if the computation by P on n does and, if so, the outputs are the same. This approach is compatible with infinite initial tape contents; for instance reserve every second tape cell for the input and store the program in the remaining cells.

Generally, one need not worry too much about arranging data on the tape and can imagine the machine having access to a fixed finite number of work tapes with independent read-write heads: Simply use every $2k$-th cell to store the information of the k-th tape and use the $2k+1$-th cells to keep track of the virtual read-write heads. In fact, via some clever coding such as Gödel pairing, one can accommodate ω-many tapes, although this can never be appreciated by classic, finitary Turing computations. The infinitary versions introduced below, however, may make use of this.

1.2.3 The Blum-Shub-Smale machine

There are different approaches to model computational reasoning on the real numbers $\mathbb{R}$.

Blum-Shub-Smale (BSS) machines provide an algebraic approach to algorithmic reasoning on the reals. The idea by Blum, Shub, and Smale bears resemblance to dynamical systems: A point in $\mathbb{R}^n$ is moved around step by step via 'elementary' operations. The elementary operations in their model are given by a program of finitely many polynomial or rational functions as instructions. Conditional jumps inside the program happen depending on certain rational functions taking positive or negative values [BSS89][1].

In contrast to BSS machines, which considers reals as atomic elements, *computable analysis (c.a.)* treats reals as limits of their finite decimal approximations [Wei00]. Turing machines are employed to approximate their computable functions: A function is computable if there is a Turing machine such that for any desired output precision the Turing machine determines the output of the function up to this precision in finite time. More formally, one considers so-called *type-2 Turing machines*: Such a machine has one input tape, one output tape, and a finite number of scratch tapes. Both the input and the output tape's heads may only move to the right. The input tape is read only, the output tape write only. The scratch tapes, however, may use the full functionality of Turing machines. The desired program now produces the output bit per bit on the output tape and may never revise a bit once written. In the nontrivial case of an output with infinite binary expansion, the machine will run for ω many steps, i.e., not terminate in the classical sense. However, it is clear that every output precision is reached at some finite time and, at any such time, only a certain number of input bits will have been read. Due to the in theory infinite runtime, this defines a model of hypercomputation. It is, however, much weaker than the hypercomputational Turing models usually considered in ordinal computability and is, obviously, easily simulated by, e.g., an ITTM (infinite time Turing machine, see below).

The main paradigmatic difference between the c.a. and the BSS approach appears to be whether computation on reals is interpreted from a numerical standpoint or from an algebraic/analytical one. There are functions that are computable in c.a. but not with BSS machines. The exponential function is easily approximated in c.a., but cannot be BSS-computed by finitely many ring operations on the input. The definition via its power series is a prime example

[1]The elementary computational steps in BSS computations are based on the operations and relations of an ordered ring. Note that the BSS definitions may be considered on arbitrary ordered rings. The ring $\mathbb{Z}$ yields the familiar Turing-equivalent computability.

of elementary use of limits of Cauchy sequences in analysis.

$$e^x = \underbrace{\sum_{k=0}^{\omega}}_{\text{infinitary device}} \underbrace{\frac{x^k}{k!}}_{\text{unproblematic since finitary}}$$

It appears quite natural to ask how this reasoning can be embedded into the computational framework provided by the BSS model. In many previous successful models of ordinal computation, lim inf's play an important role to define machine configurations at limit times. In the real analysis context, however, one could hope for an interesting theory built around strict limits, seeing them play such an integral part in analytical reasoning and the structure of $\mathbb{R}$ itself.

1.3 Ordinal computability — A brief overview

1.3.1 Infinite time Turing machines

It was Joel Hamkins' and Andy Lewis' paper on *infinite time Turing machines (ITTMs)* [HL00] that started the young field of ordinal computability in the year 2000. In the development of generalized recursion theory in the 1960's, machine models for hypercomputation have been considered[2], but in all published work that is known to the author, preference has been given to an approach via equation calculi and definability. What contrasts ordinal computability against these approaches is the focus laid on elementary computational steps, accepting all the interesting anomalies this can produce. A very good example is the ITTM, where an asymmetry between space and time is generated:

The Turing tape retains its length of ω, but the machine may carry out more than ω many steps by the addition of a limit rule: At limit times, every individual cell's content is set to the inferior limit over its previous contents (or, equivalently, superior limit as in the original publication [HL00]), the head is reset to position 0 and a dedicated limit state is entered. The resulting theory is rich and has produced concepts followed up in further ordinal computability theory: The notion of clockable ordinals for instance, i.e., those ordinals that appear as halting times. Or the question of (hyper-)decidable sets of reals, which take interesting forms whereas, in the classical finite Turing case, these sets are very easily characterized as the clopen sets. In this thesis, both these questions are investigated for other machine models, namely infinite time Blum-Shub-Smale machines (Part III) and ordinal Turing machines (Part II) respectively.

1.3.2 Ordinal Turing machines

The asymmetry of ITTMs is removed in the definition of *ordinal Turing machines (OTMs)*, which increase the tape length to Ord, so any ordinal number may appear as a cell index. As defined by Peter Koepke in [Koe05], a dedicated limit state is avoided, instead at limits, the machine is set to the least state that is assumed cofinally often. The head is not reset to 0 at limits, but instead set to the limes inferior of all previous head positions, enabling it to access the transfinitely indexed cells. The ITTM question of decidable reals (i.e., subsets

[2]see, e.g., an abstract by Azriel Levy [Lev63]

of ω) may be generalized to the question of decidable sets of ordinals for OTMs. It is easy to see that every OTM decidable set of ordinals is an element of the constructible universe L. Conversely, for every constructible subset $A \subseteq \mathrm{Ord}$ there are finitely many ordinal parameters and an OTM program P such that P with those parameters computes membership of A. In particular, bounded truth in L is computable by such machines, so OTMs may be considered a computability theoretic approach to the constructible hierarchy [Koe05].

If both time and space is limited to some admissible ordinal α, the resulting *α-Turing machines* show strong connections to admissible set theory and α-recursion theory. This theory was started in [KS09] and, with considerable improvements, continued in [Daw09].

1.3.3 Infinite time register machines

A couple of models of *infinite time register machines (ITRMs)* have been studied. Common to all is that the registers contain natural numbers and that time is, like with ITTMs, unlimited in the ordinals. They employ a $\liminf$-rule that sets registers to the $\liminf$ over previous register contents; here, however, we run into problems if the inferior limit does not take a value in the natural numbers due to the register being increased unboundedly below some limit in time. One definition has the computation break down without result if that happens [Koe06], the other resets the register to zero [KM08]. The latter approach was fleshed out along the lines of Hamkins' and Lewis' ITTM theory in [CFK+10].

1.3.4 Ordinal register machines

Ordinal register machines (ORMs) were introduced in [KS06] and have proven to be equivalent in strength to OTMs. They employ finitely many registers, each containing an ordinal, and a $\liminf$-rule. They have successfully been used to do some fine structure theory of the constructible hierarchy L by means of Silver machines [Wec10].

1.4 This thesis

We shall explore ordinal computability further, in three different areas: Firstly, we introduce an ordinal λ-calculus. In his Diploma thesis [Fis10], Tim Fischbach compared OTMs and ORMs with existing higher recursion schemes, such as Kripke's equation calculus, to constructibility and showed the equivalence of all approaches on admissible ordinals. This lifting of the most common approaches to computability theory is clearly missing a λ-calculus variant, which we shall define here. We prove its equivalence to the existing model, further strengthening Fischbach's idea of an ordinal Church-Turing thesis.

The second part consists of Chapters 3 and Chapter 4, the latter which has already been published as [SS12]. It studies the descriptive set theory of OTMs. Up to now, OTMs have been used to study the computable subsets of the ordinals. We restrict the machine to inputs of length only ω and prove that the resulting model computes exactly the Δ^1_2 sets of reals in Chapter 3. The main ingredient of this proof is Shoenfield absoluteness of Σ^1_2 sets. This result, together with several others from elementary descriptive set theory, has a strong computational flavour. In fact, we will give a proof entirely stated in the OTM formalism in Chapter 4. We then go on to study new tree representations and new pointclasses defined by OTM algorithms and establish their basic properties. Along the way, it is established that our restricted version of OTMs have a natural nondeterministic variant that behaves very much like the finite version.

Finally, after a brief introduction in Chapter 5, we address to the evident fact that Blum-Shub-Smale computablity lacks, from an analytical standpoint, the important concept of limits. In Chapter 6, which has been published as [KS12], we equip BSS machines with a suitable limit rule. Here, in contrast to other branches of ordinal computability where one deals with limits of natural or ordinal numbers, it makes sense to use strict continuous limits, as opposed to inferior or superior limits. We investigate the supremum of runtimes of these machines and compare them in strength to the Turing-style models. The extent of the computable reals of this machine is conjectured (in the mean time, this conjecture has been proved by Peter Koepke and Andrey Morozov and is awaiting publication).

1.5 Acknowledgments

I would like to express my thanks to everybody who provided support, friendship, and scientific advice in the good four years it has taken this thesis to be completed. This includes:

My colleagues at my occupation of three years as study coordinator for the Bachelor and Master programs in mathematics in Bonn, namely Dorothee Maubach, Jip Veldman, Dagmar Clos, Stephanie Fischer, and Dr. Antje Kiesel, for providing the best work environment possible and being most capable a team.

My office neighbors at the Hausdorff Center for Mathematics, Dr. David Schrittesser and Thilo Weinert, for always taking their time for discussions, sometimes spontaneous, other times in-depth, and more often than not both at the same time. I greatly enjoyed this regular source of inspiration.

Dr. Michael Meier, who continues to support me and my career, also, but not only, by providing me with the opportunity to work at the Hausdorff Center for Mathematics for the past year. His advice was and is of inestimable value to me.

My mother Katharina Seyfferth and my friends Christoph Schwerdtfeger and Felix Schoroth, on all of whom I could always rely on.

My co-authors Dr. Philipp Schlicht [SS12] and Tim Fischbach [not yet published], for the productive collaborations that form the basis for two of the three parts of my thesis.

And finally, my advisor Professor Dr. Peter Koepke, who, since my days as a Diploma student to the present day, has provided my with great opportunities, advice, and support. His encouragement and perspective were invaluable in the endeavor that was my PhD project.

Part I

Ordinal λ-calculus

Chapter 2

An ordinal λ-calculus

We are going to generalize classical λ-calculus to the ordinal domain. Our reasoning is centered around a generalization of Church numerals, i.e., terms that define the n-fold application of their first argument to the second, to numerals for transfinite ordinals. Once the new class of ordinal λ-terms is established, we define a transfinite procedure to assign to a given ordinal λ-term a normal form if one exists. This normal form procedure is compatible with the classical case, i.e., will find normal forms for classical terms whenever they exist. We go on to prove a confluency property for our procedure ('weak ordinal Church-Rosser Theorem'). The calculus thus defined is tied into the existing framework of ordinal computability: Using our terms to define a class of functions on the ordinals, we show that this class is identical with the class of $\mathbf{\Sigma}_1(L)$ definable functions on Ord.

2.1 Notation from classical λ-calculus

As a reference on classical λ-calculus the author used the monograph by Barendregt [Bar81]. Terms in classical λ-calculus are formed over the alphabet $\{\lambda, .,), (\} \cup \{v_k \mid k \in \omega\}$ by the following rules (we allow lower case letters to stand in for variables such as v_k):

(1) Every variable x is a term.

(2) If M is a term and x is a variable, then $\lambda x.M$ is a term.

(3) If M and N are terms, then so is (M, N).

We abbreviate terms of the form $\lambda x.\lambda y.M$ as $\lambda xy.M$. The *subterm* relation $S \subseteq T$ is the transitive closure of the relation $M \subseteq M$, $M \subseteq \lambda x.M$, and $M, N \subseteq (M, N)$. With respect to the expression $\lambda x.$, the notion of x as a *bound* or *free* variable has its intended meaning.

We want to identify terms that arise from each other by renaming of bound variables. By $M\frac{N}{x}$ we denote the syntactic substitution of every occurrence of the free variable x in M by the term N. If we want to replace a specific occurrence of a subterm S of T by some term R, we write the result as $T\left[\begin{smallmatrix}R\\S\end{smallmatrix}\right]$. Whenever we write such a substitution or replacement, adequate renaming of bound variables is implied to avoid variable conflicts.

The implied interpretation of λ-terms is the following. A term $(\lambda x.M, N)$ is to be interpreted as 'the application of the function $M(x)$ to N'. Accordingly, a *rule of conversion* is defined. The above term may be transformed into $M\frac{N}{x}$. More generally, for any term T, any subterm of T of the form $(\lambda x.M, N)$ may be replaced by $M\frac{N}{x}$, i.e., we may transform T into $T\left[\begin{smallmatrix}(\lambda x.M,N)\\M\frac{N}{x}\end{smallmatrix}\right]$. We call such a transformation an application of *β-conversion* or of the *β-rule* on T. A subterm of T of the form $(\lambda x.M, N)$ is called a *redex (reducible expression)* of T.

A term S is in *normal form*, if β-conversion cannot be applied to it. S is *a* normal form *of* some term T, if T is in normal form and can be obtained from S by possibly repeated applications of the β-rule. There are terms without normal forms, e.g., $(\lambda x.(x, x), \lambda x.(x, x))$. The classical theory proves that the normal form of a term is uniquely determined if it exists and can be found by a certain pattern of applications of β-conversion. This was first proved in [CR36]. We denote the normal form of a term T as $\overline{T}$.

The calculus can be fitted with various semantics. From the perspective of computability theory, maybe one of the most important ones is the λ-definability of functions on the natural numbers. There are several ways of modeling natural numbers as λ-terms; consider the following:

$$\begin{aligned}
\underline{\underline{0}} &= \lambda f x . x \\
\underline{\underline{1}} &= \lambda f x . (f, x) \\
&\vdots \\
\underline{\underline{n}} &= \lambda f x . \underbrace{(f, (f, (\ldots (f}_{n\text{-times}}, x) \ldots) \\
&\vdots
\end{aligned}$$

The terms thus defined are referred to as *Church numerals.*

A partial function $f : \mathbb{N} \supset \operatorname{dom} f \to \mathbb{N}$ is called *λ-definable* if there is a λ-term F such that for all $n \in \operatorname{dom} f$:

$$\overline{(F, \underline{\underline{n}})} = \underline{\underline{f(n)}}$$

The class of λ-definable functions is identical to the class of Turing-computable functions. By virtue of the Church–Turing thesis we also speak of the class of computable functions.

2.2 λI-calculus

The basis for our generalization of λ-calculus shall be given by the $\lambda\mathbf{I}$-calculus as described in [Bar81, Chapter 9]. The $\lambda\mathbf{I}$-terms form a subset of the λ-terms and are formed by replacing formation rule (2) by the following:

(2) If M is a term and x is a variable that appears free in M, then $\lambda x.M$ is a term.

With $\lambda\mathbf{I}$, trivial applications ('forget the argument') are impossible, as terms of the form $\mathbf{K} = \lambda xy.x$ are illegal. So, in general, case distinctions (returning one of several arguments depending on the situation) or constant functions cannot be defined in $\lambda\mathbf{I}$. For functions on numerals, however, this can be circumvented, exploiting the syntactic structure of Church numerals.

Definition 2.1 ([Bar81])**.** Set $\mathbf{I} = \lambda y.y$. This is a $\lambda\mathbf{I}$-term defining the unary identity function.

As an example, the numeral $\underline{\underline{0}}$ could be replaced by $\underline{\underline{0'}} = \lambda f x.(((f, \mathbf{I}), \mathbf{I}), x)$. Let $\underline{\underline{n'}} = \underline{\underline{n}}$ for all $n > 0$. Note that the normal form of $((\underline{\underline{0'}}, \underline{\underline{n'}}), \underline{\underline{m'}})$ is $\underline{\underline{m'}}$, so, on numerals, the meaning '0-fold application of the first argument to the second' is retained.

In contrast to $\lambda\mathbf{I}$-terms, the full set of λ-terms is sometimes referred to as $\lambda\mathbf{K}$-terms. Omitting further details which can be found in [Bar81], we state the following:

Fact 2.2 ([Bar81])**.** *The* $\lambda\mathbf{I}$*-definable functions on natural numbers coincide with the* $\lambda\mathbf{K}$*-definable ones.* □

2.3 Ordinal λ-terms

Our approach revolves around the idea of generalizing Church numerals from 'the n-fold application of f to x' to 'the α-fold application of f to x'. The intent behind that idea is that terms defining the successor function or arithmetic should generalize to the successor function on ordinals or ordinal arithmetic.

Note 2.3. In developing our theory, we briefly considered introducing terms of transfinite length, but the asymmetry of ordinals — a limit ordinal has a right neighbor (a least larger ordinal) but no left neighbor (largest smaller ordinal) — limits the intuitive use of syntactical operations on such strings. Instead, we introduce symbols for ordinals on term level and we propose the following generalization of λ-terms to ordinal λ-terms.

Definition 2.4 (Fischbach-Seyfferth)**.** Over the alphabet $\Sigma_{\mathrm{Ord}} = \{\lambda, .,), (\} \cup \{v_k \mid k \in \omega\} \cup \{{}^{\alpha} \mid \alpha \in \mathrm{Ord}\}$ define the set $\mathrm{Term}_{\mathrm{Ord}}$ of *ordinal λ-terms* by

(1) Every variable x is an ordinal λ-term.

(2) If M is an ordinal λ-term and x appears free in M, then $\lambda x.M$ is an ordinal λ-term.

(3) If α is an ordinal and M and N are ordinal λ-terms, then so is ${}^{\alpha}(M, N)$.

We often write (M, N) instead of ${}^{1}(M, N)$.

Informally, we refer to ordinal λ-terms just as *terms*.

We introduce an equivalence relation $\simeq_v$ on terms, identifying all terms that can be obtained from each other by renaming of bound variables. If V is a finite set of variables with largest element v_i, define for every equivalence its *v-minimal term over* V as the one where all bound variables are named v_{i+1}, v_{i+2}, v_{i+3}, etc. from left to right. For a term T, we denote by T_v^V its v-minimal term over V. We simply write v-minimal and T_v if $V = \emptyset$.

Definition 2.5 (Fischbach-Seyfferth)**.** The *(ordinal) Church numerals* take the form $\underline{\alpha} = \lambda fx.{}^{\alpha}(f, x)$ for $\alpha \in \mathrm{Ord}$. More generally, we refer to all terms $\simeq_v$-equivalent to some $\underline{\alpha}$ as Church numerals.

The intended meaning of terms like ${}^{\beta}(M, {}^{\alpha}(M, N))$ and ${}^{\alpha+\beta}(M, N)$ is the same. So, we want to identify all the terms of the form

$$^{\alpha_{k-1}}(M, {}^{\alpha_{k-2}}(M, \ldots {}^{\alpha_0}(M, N) \ldots))$$

with

$$^{\alpha_0+\alpha_1+\ldots+\alpha_{k-1}}(M, N).$$

Let T be a term. We call a replacement of all subterms of T that are of the former form by their equivalent terms of the latter form a *contraction of applications of* T. We define an equivalence relation $\simeq_a$ by identifying every term T with the terms resulting from contractions of its applications and closing transitively. For a given term T we define its *a-minimal term* T_a as the shortest term a-equivalent to T.

In order to define an equivalent to the β-normal form, we define a transfinite procedure that for every term either finds a term we shall call its normal form or diverges, the latter which we will interpret as the term not having a normal form.

2.4 Normal form derivation

The normal form derivation will be a transfinite procedure. We declare a form of limit convergence for our ordinal λ-terms.

Definition 2.6 (Fischbach-Seyfferth)**.** Consider a term M as a finite sequence of symbols in Σ_{Ord}, i.e., $M : n \to \Sigma_{\mathrm{Ord}}$ for some natural number $n \in \omega$. Let $(j_i \mid 0 \leq i < k)$ be the increasing sequence of those $j < n$ with $M(j) \in \{ {}^{\alpha} \mid \alpha \in \mathrm{Ord}\}$. Let $\vec{\alpha} = (\alpha_0, \dots, \alpha_{k-1})$ be some sequence of ordinals. Define

(i) the *flesh* of M as $\mathrm{fl}(M) = (M(j_i) \mid i < k)$

(ii) the *skeleton* of M as $\mathrm{sk}(M) : n \to \mathrm{ran}(M)$ with

$$\mathrm{sk}(M)(j) := \begin{cases} M(j), & \text{if } j \notin \{j_i \mid 0 \leq i < k\} \\ {}^1, & \text{if } j \in \{j_i \mid 0 \leq i < k\}. \end{cases}$$

(iii) the *insertion of $\vec{\alpha}$ in M* as

$$M[\vec{\alpha}] = \begin{cases} M(j), & \text{if } j \notin \{j_i \mid 0 \leq i < k\} \\ {}^{\alpha_i}, & \text{if } j = j_i \text{ for some } i < k. \end{cases}$$

Note that $\mathrm{sk}(M)[\mathrm{fl}(M)] = M$.

Definition 2.7 (Fischbach-Seyfferth)**.** Let α be a limit ordinal.

(i) Let $s : \alpha \to \mathrm{Ord}^n$ be a sequence of n-tuples of ordinals for some $n < \omega$. Define the *pointwise limes inferior* by

$$\liminf_{\beta \to \alpha} s(\beta) := (\liminf_{\beta \to \alpha} s(\beta)_0, \dots, \liminf_{\beta \to \alpha} s(\beta)_{n-1}).$$

(ii) Let $s : \alpha \to \mathrm{Term}_{\mathrm{Ord}}$ be a sequence of terms. We say that *the skeletons of s converge*, if there is a a-minimal and v-minimal skeleton S such that there is a $\beta < \alpha$ and for all $\beta < \gamma < \alpha$ we have $\mathrm{sk}(s(\gamma))_a \simeq_v S$. We will call S the *limit skeleton for s.* If V is a finite set of variables, S may be chosen v-minimal over V; we then speak of the limit skeleton *over V*.

(iii) Let $s : \alpha \to \Sigma^*_{\mathrm{Ord}}$ be a sequence of terms whose skeletons converge to limit skeleton S. Let γ be minimal such that $\mathrm{sk}(s(\beta)_a) = S$ for $\gamma < \beta < \alpha$. Then the *syntactical limes inferior* of s exists and is defined by

$$\liminf_{\beta \to \alpha} s(\beta) := \mathrm{sk}(S)[\liminf_{\beta \to \alpha, \beta > \gamma} \mathrm{fl}(s(\beta)_a)]$$

If S is the limit skeleton over some finite set of variables V, we speak of the syntactical limes inferior over V, written $\liminf^V_{\beta \to \alpha} s(\beta)$

Note 2.8. In the following we give a deterministic procedure to arrive a given ordinal λ-term's normal form. Every step can be seen to correspond to one application of the classical β-rule. In the classical $\lambda\mathbf{I}$-calculus, any pattern of iterated application of the β-rule eventually yields a normal form. The idea of α-fold application of one term to another implies a transfinite length of applications of a β-rule, and we want to make use of the limit notions for terms we just defined. We chose to give up the nondeterministic freedom of the finite case to produce stabilizing and natural behavior at limits. In turn, we get some of that freedom back by proving a weak confluency property in Section 2.5. There, we also conjecture a stronger property that would enable us to perform finitely many arbitrary deviations from the algorithm.

Before we give a rigorous definition, it might be helpful to look at the process for arriving at a normal form we have in mind a bit more graphically. The algorithm will maintain a stack. Each stack element is a term whose normal form is to be determined. The bottom element of the stack is the original term T we wish to reduce to its normal form by some generalizations of the β-rule. The next element shall be the *leftmost redex* of T, i.e., a leftmost subterm S of the form $S = {}^{\alpha}(\lambda x.M, N)$ where $\alpha > 0$. A redex is called leftmost if its operating λ, i.e., the λ that is spelled out in the representation of S above, appears to the left of all other operating λ's of redexes of T. So, S is put on the second stack level. Recursively, the algorithm will determine the normal form of S. In the mean time, the stack will get built up and torn down again and as soon as S's normal form $\overline{S}$ is found, our stack will contain exactly two elements: T as the bottom one with $\overline{S}$ on top. The next step will be to remove $\overline{S}$ from the stack, replace S in T with $\overline{S}$ and start the procedure over for the resulting term. Eventually, the bottom element will contain no more redexes and a normal form is found.

So how does the algorithm proceed to determine the normal form of some redex $S = {}^{\alpha}(\lambda x.M, N)$? We retain the intuition of 'the α-fold application of M to N' by the following procedure: Determine, by putting on the stack consecutively, the normal forms of the approximations ${}^{1}(\lambda x.M, N)$, ${}^{2}(\lambda x.M, N)$, etc. At limit times, syntactical inferior limits are taken (if they exist, otherwise the normal form procedure breaks down). More precisely, instead of ${}^{\gamma+1}(\lambda x.M, N)$, we evaluate the a-equivalent term $(\lambda x.M, {}^{\gamma}(\lambda x.M, N))$, substituting the term N' we determined in the previous steps as the normal form of ${}^{\gamma}(\lambda x.M, N)$, which in the end gives us $(\lambda x.M, N')$ to evaluate. We can now rely on an application of what is know as the β-rule in the finite case to end up with $M\frac{N'}{x}$ which is what is being put on the stack instead of ${}^{\gamma+1}(\lambda x.M, N)$.

Finally, we have to deal with our stack length becoming infinite. This might happen via the use of terms that work like $(\lambda x.(x, x), \lambda x.(x, x))$ and may be used as so-called fixed-point combinators in recursive definitions. For instance, one usually implements unbounded search by a term describing the following function $Q(\alpha)$: 'if condition P holds on α then return α, else evaluate and return $Q(\alpha + 1)$'. A normal form procedure for $Q(0)$ will build up a stack of height β if β is the least ordinal such that P holds. If the stack length approaches a limit ξ, we shall define the stack content at level ξ in the following way: First, identify the first (from the bottom) term on the stack whose skeleton appears cofinally often below ξ as skeleton of terms on the stack. Now, set as the term on level ξ the syntactical lim inf over all terms on the stack with this skeleton. In the above example, we will 'try α' (i.e., put $Q(\alpha)$ on the stack), then do some steps to determine whether P holds for α and if not 'try $\alpha+1$'. In the next limit we want to 'try the first limit after α', i.e., put $Q(\alpha+\omega)$ on the stack. The term $Q(0)$ representing 'try 0' obviously is the first whose skeleton appears cofinally often. Theorem 2.25 will confirm this behavior. To avoid problems with backtracking downwards in the ordinals, we will keep track of the stack height on which the first term that lends its skeleton to the limit appears. That way, as soon as a normal form is found (in our example the least β such that P holds) and has been handed down step by step until the stack is torn down to some limit height, we can propagate this normal form directly downwards to whenever $Q(0)$ was put on the stack.

Note 2.9. Terms of the form ${}^0(\lambda x.M, N)$ are not treated as reducible; they are unchanged by the algorithm save for internal modifications of M and N and may vanish only through contractions of applications. On could argue that the intended meaning behind such a term is simply N (the 0-fold application of M to N) but such transformations would reintroduce terms of the form $\lambda xy.x$, violating the boundaries of $\lambda\mathbf{I}$-calculus.

For our purposes, there is an added benefit of not resolving 0-fold applications: The numeral $\underline{0} = \lambda fx.{}^0(f,x)$ is of the same syntactical form as the other numerals, enabling $\underline{0}$ to be a possible value of a syntactical $\liminf$ of numerals. However, not setting $\underline{0}$ apart from the other numerals introduces a difficulty with arithmetic: Several classical algorithms for arithmetic (predecessor of natural numbers, subtraction, etc.) rely heavily on a term that recognizes the numeral for 0 among all the other numerals. We resolve this issue by expanding our calculus by a term that defines equality on ordinals in Definition 2.12.

For the following rigorous definition, some additional information is coded into the stack elements, but the above structure remains valid. In general, we consider a stack to be a sequence indexed by a successor ordinal. Operations changing the stack are either restricting the sequence to another successor ordinal or adding a new element on top. The algorithm below will define the behavior when infinitely many end extensions are carried out. The stack elements will be tuples $(\cdot)$, so we write a stack of tuples as $\langle(\cdot),(\cdot),\dots,(\cdot)\rangle$. We use the symbol $\sqcup$ to denote the composition of two stacks.

We need to elaborate how renaming of bound variables will be handled in the normal form procedure. Whenever a substitution of the form $M\frac{N}{x}$ or a replacement of the form $M\left[\begin{smallmatrix}N\\P\end{smallmatrix}\right]$ is carried out, we rename bound variables adequately: In every renaming, new variables are chosen as to not accidentally bind a free variable of some term farther down on the stack. At limits, we chose the limit terms v-minimally, while avoiding the variables of certain lower stack levels: If all stack levels are finite, we can certainly avoid all of the only finitely many variables used below. If an infinite stack level is reached, we only require the variables up to the level from which the skeleton is lent to the limit to be avoided. Inductively, at any given point in time we only need to avoid finitely many variables.

Definition 2.10 (Fischbach-Seyfferth)**.** Let N be a term. A function $s:\theta\to(\Sigma^*_{\mathrm{Ord}}\times\Sigma^*_{\mathrm{Ord}}\times\mathrm{Ord}\times\mathrm{Ord}\times\mathrm{Ord})^{<\mathrm{Ord}}$ is called a *normal form derivation of N* if it satisfies the following conditions:

(a) $s(0)=\langle(N,\emptyset,0,0,0)\rangle$.

(b) Let $\nu < \theta$.

   (i) If the topmost element of $s(\nu)$ is of the form $(N', \emptyset, 0, 0, \delta)$, and N' is not a-minimal, and $R = {}^{\beta}(P, {}^{\alpha}(P, Q))$ be the leftmost subterm on which a contraction of applications may be carried out, then $\nu + 1 < \theta$ and $s(\nu + 1) = s(\nu) \restriction \nu \sqcup \langle (N\left[{}_{\alpha+\beta}\overset{R}{(P,Q)}\right], \emptyset, 0, 0, \delta)\rangle$.

   (ii) If the topmost element of $s(\nu)$ is of the form $(N', \emptyset, 0, 0, \delta)$, and N' is a-minimal and contains a subterm of the form ${}^{\gamma}(\lambda x.P, Q)$ where $\gamma > 0$, and ${}^{\alpha}(\lambda x.M, N'')$ is the leftmost such subterm, then $\nu + 1 < \theta$ and $s(\nu + 1) = s(\nu) \sqcup \langle (N'', \lambda x.M, \alpha, 0, \delta)\rangle$.

   (iii) If the topmost element of $s(\nu)$ is of the form $(N'', \lambda x.M, \alpha, \beta, \delta)$ with $\beta < \alpha$, then $\nu + 1 < \theta$ and $s(\nu + 1) = s(\nu) \sqcup \langle (M\frac{N''}{x}, \emptyset, 0, 0, \delta)\rangle$.

   (iv) If the two topmost elements of $s(\nu)$ are of the form $\langle (N'', \lambda x.M, \alpha, \beta, \delta), (N', \emptyset, 0, 0, \delta)\rangle$ and N' is a-minimal and does not have a subterm of the form ${}^{\gamma}(\lambda x.P, Q)$ with $\gamma > 0$, then $\nu + 1 < \theta$ and $s(\nu + 1) = s(\nu) \restriction (|s(\nu)| - 2) \sqcup \langle (N', \lambda x.M, \alpha, \beta + 1, \delta)\rangle$.

   (v) If $|s(\nu)|$ is successor of a limit and the topmost element of $s(\nu)$ is of the form $(N', \emptyset, 0, 0, \delta)$ and N' is a-minimal and does not have a subterm of the form ${}^{\gamma}(\lambda x.P, Q)$ with $\gamma > 0$ and $s(\nu)(\delta) = (N'', \emptyset, 0, 0, \gamma)$, then $s(\nu + 1) = s(\nu) \restriction \delta \sqcup \langle (N', \emptyset, 0, 0, \gamma)\rangle$.

   (vi) If the two topmost elements of $s(\nu)$ are of the form $\langle (N', \emptyset, 0, 0, \delta), (N'', \lambda x.M, \alpha, \alpha, \delta)\rangle$, then $\nu + 1 < \theta$ and $s(\nu) \restriction (n - 2) \sqcup \langle (\tilde{N}, \emptyset, 0, 0, \delta)\rangle$, where $\tilde{N}$ arises from N' by replacing the leftmost subterm of the form ${}^{\gamma}(\lambda x.P, Q)$ with N''.

   (vii) If none of the above conditions hold, apparently we have $s(\nu) = (N', \emptyset, 0, 0, 0)$, N' is a-minimal and does not have any redexes, and $|s(\nu)| = 1$. Then $\theta = \nu + 1$ and N' is called *the result of the normal form derivation of* N, written $\overline{N} = N'$.

(c) Let ξ be a limit ordinal such that $s \restriction \xi$ is defined.

   (i) If $\liminf_{\nu<\xi} |s(\nu)| = \gamma$ and the sequence $(\nu)_{\nu<\xi \wedge |s(\nu)|=\gamma}$ is unbounded in ξ, note that the stacks $(s(\nu) \restriction \gamma)_{\nu<\xi\wedge|s(\nu)|=\gamma}$ are eventually constant some stack s of length γ. Let V be the set of variables of the terms on the stack levels up to γ. If the skeletons of $(s(\nu)(\gamma))_{\nu<\xi\wedge|s(\nu)|=\gamma}$ converge, then

$$s(\xi) = s \sqcup \Big\langle \Big(\liminf{}^{V}_{\nu<\xi\wedge|s(\nu)|=\gamma} \mathrm{pr}_0(s(\nu)), \\ \emptyset, 0, 0, \liminf{}_{\nu<\xi\wedge|s(\nu)|=\gamma} \mathrm{pr}_4(s(\nu))\Big)\Big\rangle .$$

   If they do not converge, then $\xi = \theta$ and the normal form derivation of N is said to *diverge*, written $\overline{N}\uparrow$.

   (ii) If, on the other hand, $\liminf_{\nu<\xi} |s(\nu)| = \gamma$, η is a limit, and $(|s(\nu)|)_{\nu<\xi}$ is unbounded in η, we require that for every $\gamma < \eta$ there is a time $\nu < \xi$ such that for every $\mu > \nu$ we have $s(\mu) \restriction \gamma = s(\nu) \restriction \gamma$. This way we obtain a *limit stack* t of length η. It remains to be determined what element is to be put on top of this limit stack at time ξ. Choose the first (i.e., with minimal γ) skeleton $u = \mathrm{sk}\,\mathrm{pr}_0(t(\gamma))$ of

$(\mathrm{pr}_0(t(\nu)))_{\nu<\eta\wedge\text{``}\nu\text{ is even''}}$ such that either u or an v- or a-equivalent term appears unboundedly often as skeleton in the first coordinate of t. Let V be the set of variables of the terms on the stack levels up to γ. Set

$$s(\xi) = t \sqcup \left\langle \left(\liminf{}^{V}_{\nu<\eta\wedge\text{``}\nu\text{ is even''}\wedge \mathrm{sk}\,\mathrm{pr}_0(t(\nu))=u} \mathrm{pr}_0(t(\nu)), \emptyset, 0, 0, \gamma\right)\right\rangle.$$

If no limit stack t exists or no skeleton appears unboundedly often in t, then $\xi = \theta$ and the normal form derivation of N is said to *diverge*, written $\overline{N}\uparrow$.

If no normal form derivation exists for some term N, we also say that the normal form derivation of N *diverges*, written $\overline{N}\uparrow$.

Definition 2.11 (Seyfferth)**.** Let $\alpha_0, \dots, \alpha_n$ be a finite sequence of ordinal numbers. A function $f : \mathrm{Ord}^k \to \mathrm{Ord}$ is *ordinal λ-definable in parameters $\vec{\alpha}$* if there is an ordinal λ-term T in which all applications are of the form ${}^{\beta}(\cdot,\cdot)$ where $\beta \in \omega \cup \{\alpha_0, \dots, \alpha_{n-1}\}$ such that for all $(\gamma_0, \dots, \gamma_{k-1}) \in \mathrm{Ord}^k$ we have

$$\overline{(\dots(T, \underline{\gamma_0}), \underline{\gamma_1}), \dots), \underline{\gamma_{k-1}})} \simeq_v \underline{f(\gamma_0, \dots, \gamma_{k-1})}.$$

If f is a partial function, we call f ordinal λ-definable in $\vec{\alpha}$ if $f \restriction \mathrm{dom}\, f$ is ordinal λ-definable in $\vec{\alpha}$ and $\overline{(T, \underline{\gamma})}\uparrow$ on $\gamma \notin \mathrm{dom}\, f$. If $\vec{\alpha} = \emptyset$, we simply speak of *ordinal λ-definability.*

As explained in Note 2.9, we would like to add the capability of defining equality on ordinals to our calculus:

Definition 2.12 (Seyfferth)**.** Let us add a constant symbol E to our alphabet and consider the terms formed over $\Sigma_{\mathrm{Ord}} \cup \{E\}$ with the additional rule 'E is a term' as the *ordinal $\lambda + E$-terms.* We extend Definition 2.10 by a case for terms of the form ${}^1({}^1(E, \underline{\alpha}), \underline{\beta})$: The algorithm is to replace ${}^1({}^1(E, \underline{\alpha}), \underline{\beta})$ with the normal form $\mathbf{T}_I = \lambda xy.(((y, \mathbf{I}), \mathbf{I}), x)$ if $\alpha = \beta$ and with the normal form $\mathbf{F}_I = \lambda x.(((x, \mathbf{I}), \mathbf{I}), \mathbf{I})$ else. In all other cases, E is to be treated like a variable symbol. The resulting notion of definability for functions on the ordinals is that of *ordinal $\lambda + E$-definable* functions.

The terms $\mathbf{T}_I$ and $\mathbf{F}_I$ will be used in the following manner:

$$\text{Suppose } \overline{((P, \mathbf{I}), \mathbf{I})} \simeq_v \overline{((Q, \mathbf{I}), \mathbf{I})} \simeq_v \mathbf{I}.$$
$$\text{Then } \overline{((B, P), Q)} \simeq_v \begin{cases} \overline{P} & \text{, if } \overline{B} = \mathbf{T}_I \\ \overline{Q} & \text{, if } \overline{B} = \mathbf{F}_I. \end{cases}$$

The term $((B, P), Q)$ hence may be read as `if` B `then` P `else` Q. We shall also consider variations of $\mathbf{T}_I$ and $\mathbf{F}_I$: The terms $\mathbf{T}_J$ and $\mathbf{F}_J$ as well as $\mathbf{T}_3$ and $\mathbf{F}_3$ are defined later on and behave similarly, for terms P and Q vanishing under different conditions.

Some examples are in order now. We take the liberty of abbreviating some terms in these examples, numerals for instance. Additional steps are added to write them out whenever necessary. We may also use extra lines to carry out substitutions etc. Apart from this, there is a line for every step in the normal form derivation where the stack has odd height, i.e., the topmost element is of the form $(N', \emptyset, 0, 0, \delta)$. Every line contains the first entry of the topmost stack element. Stack height is suggested by indentation. We mark the approximation steps by leading numbers to facilitate reading. We highlight the leftmost redex by $\overline{\text{overlining}}$. We deliberately use our normal form notation here: the algorithm will recursively determine the normal form of the overlined term and, once found, replace the term with its normal form.

Example 2.13 (Seyfferth)**.** Let us count up to ω to illustrate the desired lim inf behavior. The term $S_c^+ = \lambda nfx.(f, ((n, f), x))$ classically defines the successor function of a given numeral. We show that the normal form of ${}^{\omega}(S_c^+, \underline{0})$ is $\underline{\omega}$. Let us run our algorithm:

${}^{\omega}(S_c^+, \underline{0})$			write out S_c^+
$\overline{{}^{\omega}(\lambda nfx.(f, ((n, f), x)), \underline{0})}$			the entire term is the leftmost redex
	1:	$\lambda fx.(f, ((n, f), x))\frac{\underline{0}}{n}$	start the approximation of the ω redex
		$\lambda fx.(f, ((\underline{0}, f), x))$	write out $\underline{0}$
		$\lambda fx.(f, (\overline{(\lambda gy.{}^{0}(g, y), f)}, x))$	identify leftmost redex
		1: $\lambda y.{}^{0}(g, y)\frac{f}{g}$	first and only approximation step
		$\lambda y.{}^{0}(f, y)$	return redex-free term
		$\lambda fx.(f, \overline{(\lambda y.{}^{0}(f, y), x)})$	identify leftmost redex
		1: ${}^{0}(f, y)\frac{x}{y}$	first and only approximation step
		${}^{0}(f, x)$	return redex-free term
		$\lambda fx.(f, {}^{0}(f, x))$	contract applications to obtain an a-minimal term
		$\lambda fx.(f, x)$	
		$\underline{1}$	supply redex-free term to next approximation step
	2:	$\lambda fx.(f, ((n, f), x))\frac{\underline{1}}{n}$	second approximation step
		$\vdots$	analogously to first approximation step
		$\underline{2}$	hand redex-free term to next approximation step
	$\vdots$		analogously for all finite steps
	ω:	$\liminf_{n \to \omega} \underline{n}$	take lim inf at limit step
		$\liminf_{n \to \omega} \lambda fx.{}^{n}(f, x)$	evaluate syntactical lim inf
		$\lambda fx.{}^{\omega}(f, x)$	last approximation step; return redex-free form
$\lambda fx.{}^{\omega}(f, x)$			normal form reached
$\underline{\omega}$			

The example above can be used to prove that the successor function on ordinals is ordinal λ-definable. The following example is given to show some nested limits, suggesting that recursively defined ordinal arithmetic can be implemented in a straightforward manner.

Example 2.14 (Seyfferth)**.** If $\alpha, \beta > 0$ then $\lambda fx.((\underline{\beta}, (\underline{\alpha}, f)), x)$ defines the product $\alpha \cdot \beta$.

$\lambda fx.((\underline{\beta}, (\underline{\alpha}, f)), x)$	write out numerals
$\lambda fx.(\overline{(\lambda gy.{}^{\beta}(g,y), (\lambda hz.{}^{\alpha}(h,z), f))}, x)$	identify leftmost redex
$\quad 1: \quad \lambda y.{}^{\beta}(g,y)\dfrac{(\lambda hz.{}^{\alpha}(h,z), f)}{g}$	first and only approximation step
$\qquad \lambda y.{}^{\beta}(\overline{(\lambda hz.{}^{\alpha}(h,z), f)}, y)$	identify leftmost redex
$\qquad 1: \quad \lambda z.{}^{\alpha}(h,z)\dfrac{f}{h}$	first and only approximation step
$\qquad\qquad \lambda z.{}^{\alpha}(f,z)$	return redex free term
$\qquad \lambda y.\overline{{}^{\beta}(\lambda z.{}^{\alpha}(f,z), y)}$	identify leftmost redex
$\qquad 1: \quad {}^{\alpha}(f,z)\dfrac{y}{z}$	first approximation step of the β redex
$\qquad\qquad {}^{\alpha}(f,y)$	hand redex-free term to next approximation step
$\qquad 2: \quad {}^{\alpha}(f,z)\dfrac{{}^{\alpha}(f,y)}{z}$	second approximation step
$\qquad\qquad {}^{\alpha}(f, {}^{\alpha}(f,y))$	contract applications
$\qquad\qquad {}^{\alpha\cdot 2}(f,y)$	hand redex-free term to next approximation step
$\qquad \vdots$	analogously for all finite steps
$\qquad \omega: \quad \liminf\limits_{n\to\omega} {}^{\alpha\cdot n}(f,y)$	take lim inf at limit step
$\qquad\qquad {}^{\alpha\cdot\omega}(f,y)$	hand redex-free term to next approximation step
$\qquad \vdots$	analogously for all further steps
$\qquad \beta: \quad \liminf\limits_{\gamma\to\beta} {}^{\alpha\cdot\gamma}(f,y)$	take lim inf at limit step
$\qquad\qquad {}^{\alpha\cdot\beta}(f,y)$	last approximation step; return redex-free term
$\qquad \lambda y.{}^{\alpha\cdot\beta}(f,y)$	return redex-free term
$\lambda fx.\overline{(\lambda y.{}^{\alpha\cdot\beta}(f,y), x)}$	identify leftmost redex
$\quad 1: \quad \lambda y.{}^{\alpha\cdot\beta}(f,y)\dfrac{x}{y}$	first and only approximation step
$\qquad {}^{\alpha\cdot\beta}(f,x)$	return redex-free term
$\lambda fx.{}^{\alpha\cdot\beta}(f,x)$	normal form reached
$\underline{\alpha\cdot\beta}$	

The two examples just given only illustrate the limit case (c)(i). For an example for case (c)(ii) see section 2.6.2.

Classical $\lambda\mathbf{I}$-terms are ordinal λ-terms. On the other hand, if we have an ordinal λ-term where for all applications ${}^{\alpha}(M,N)$ we have that α is finite, we can convert it to a classical $\lambda\mathbf{I}$-term by a map ϕ given by:

- replacing any subterm of the form ${}^{n}(M,N)$ by $\underbrace{(M,(M,(\dots(M}_{n\text{-times}},N)\dots)$ if $n>0$,
- replacing any subterm of the form ${}^{0}(M,N)$ by $(((M,\mathbf{I}),\mathbf{I}),N)$. In particular, this maps $\underline{0}$ to $\underline{\underline{0}}' = \lambda fx.(((f,\mathbf{I}),\mathbf{I}),x)$, the term [Bar81] uses in the treatment of $\lambda\mathbf{I}$-calculus.

Proposition 2.15 (Seyfferth). *If M is a classical $\lambda\mathbf{I}$-term with classical normal form M' and M'' is the output of our algorithm on input M, then $\phi(M'') \simeq_v M'_a$.*

Proof. In $\lambda\mathbf{I}$-calculus, every reduction strategy (pattern of applying the β-rule to various subterms until no redex is left) is normalizing, i.e., eventually yields normal forms. Our algorithm, although working with a-minimal terms, will simply run a finite number of applications of the β-rule before halting with a term M'' without redexes. Converting this term to a classical $\lambda\mathbf{I}$-term does not introduce any redexes, so the resulting term is also classically in normal form. Since classically normal forms are unique up to renaming of variables, we have indeed found a term $\simeq_v$-equivalent to M'. □

2.5 A weak Church-Rosser Theorem for our algorithm

The classical Church-Rosser result establishes that for any two terms Q and Q' that are obtained from the same term P via β-reduction, there is a term R that can be obtained from Q and Q' by β-reduction. This 'diamond property' is known as the Church-Rosser property. It ensures that normal forms are unique. In the $\lambda\mathbf{K}$-calculus, a term's unique normal form is obtained by a certain pattern of applications of the β-rule, whereas in $\lambda\mathbf{I}$, any pattern of applications of the β-rule leads to the term's normal form, given that one exists. In our situation, where restrict ourselves from applying the β-rule freely for the sake of convergence at limits, we propose the following as the correct lifting of the Church-Rosser theorem:

Conjecture 2.16 (Seyfferth). *Let T be a term with normal form and let $S \subseteq T$ be a subterm. Then $\overline{T} \simeq_v \overline{T\left[\begin{smallmatrix}\overline{S}\\ S\end{smallmatrix}\right]}$.*

For the purposes of this thesis, the following result is sufficient, as it will establish that the composition of two λ-definable functions is λ-definable. We will refer to it as the *weak Church-Rosser Theorem for ordinal λ-calculus* (results for $\lambda + E$ follow analogously).

Theorem 2.17 (Seyfferth). *Let T be a term with normal form and let $S \subseteq T$ be a subterm of the form $S = {}^{\alpha}(M,N)$ such that all free variables in S are not bound in T. Then $\overline{T} \simeq_v \overline{T\left[\begin{smallmatrix}\overline{S}\\ S\end{smallmatrix}\right]}$.*

Proof. Fix some term S of the form ${}^{\alpha}(M, N)$. We can assume that S is not in normal form, otherwise we would be done. Let T be some term with S as a subterm, such that all free variables in S are not bound in T. Assume that T has a normal form. We will compare the normal form derivations t of T and t' of $T\left[\frac{\overline{S}}{S}\right]$. It will be evident that t and t' have basically the same steps, except for subsequences of t that parallel the normal form derivation of S.

Let θ be the length of t and let θ' be the length of t'. We define recursively an injective and weakly monotonous map $f : \theta' \to \theta$ such that for every $\gamma < \theta'$ we have that every stack element of $t'(\gamma)$ can be obtained from $t(f(\gamma))$ by replacing copies of S by copies of $\overline{S}$ and the stacks $t'(\gamma)$ and $t(\gamma)$ have the same height. We define this map 'the other way round', by defining a surjective and increasing but not injective partial map g from t to t' such that our condition holds: At any time δ, we have that all stack elements of $t'(g(\delta))$ arise from $t(\delta)$ by replacing copies of S by copies of $\overline{S}$ (and possibly some variable renaming, which we shall surpress for the rest of this argument). We then can choose as $f(\gamma)$ the largest δ with $g(\delta) = \gamma$ (it will be clear from the construction that there always will be a largest pre-image). Let $\theta_0 + 1 = \theta$ and $\theta'_0 + 1 = \theta'$ (normal form derivations always have successor length). The construction below makes sure that g is defined on θ_0. Since g is surjective and increasing, $g(\theta_0) = \theta'_0$.

Then $t'(\theta'_0)$ is a stack of height 1 and its top element is in normal form. Therefore $t(f(\theta'_0))$ is also also a stack of height 1 with top element in normal form and we have $f(\theta'_0) = \theta_0$ and $\overline{T} \simeq_v \overline{T\left[\frac{\overline{S}}{S}\right]}$.

Throughout the recursive definition of g, we shall keep track of the *residuals* of S along the normal form derivation t. The term residual was coined by Church and Rosser in their paper proving confluency properties for the classical $\lambda\mathbf{I}$-calculus [CR36]. We use it in the following way: During a normal form derivation, the subterm S of the original term T may be moved around or duplicated due to other redexes being resolved. The resulting subterms ${}^{\alpha}(M', N')$ are called residuals of S. Residuals of S vanish as soon as the application term ${}^{\alpha}(M', N')$ is put on the stack and evaluated. Note that the free variables of residuals of S remain unbound in the surrounding term. We will make only informal use of the notion of residuals, as all the details will be made clear in the definition of g.

At time 0, we set $g(0) = 0$. Our condition holds. So let g be defined up to δ and let $t(\delta)$ be of odd height. Consider the term T' that is the topmost element of $t(\delta)$ (more precisely, the first component of the topmost element, which is the term in whose normal form we are interested at time δ). Assume that our condition holds up to δ and let S_0, S_1, …, S_{k-1} denote those copies of S that are residuals of S in T'. Note that, as with S in T, all free variables of the S_i are unbound in T'.

Case 0. If T' is not a-minimal, consider the leftmost term of the form ${}^{\theta}(Q, {}^{\eta}(Q, R))$. If the second part ${}^{\eta}(Q, R)$ is not one of the S_i, set $g(\delta + 1) = g(\delta) + 1$ as in both t and t' the next step simply is the contraction of applications. If ${}^{\eta}(Q, R) = S_i$, observe that the next step in t is the contraction of applications; followed by the first η-many approximation steps, i.e., the entire normal form derivation of S_i minus the last step that returns the result; again followed by the next θ-many approximation steps. Let γ be the length of the first η-many approximation steps. Let β be the length of the next θ-many approximation steps. Set $g(\delta + 1 + \gamma + i) = g(\delta) + 1 + i$ for $i < \beta$. Then our condition holds for $t(\delta+1+\gamma)$ and $t'(g(\delta)+1)$ as in both stacks we are just at the beginning of the θ-many approximation steps of ${}^{\theta}(Q,\overline{S})$. The condition carries over to the next β-many steps that are carried out in the same fashion in t and t'.

Case 1. If T' is in normal form, then the next two steps of t will be to substitute T' into a term immediately below T' on the stack. The topmost element of $t'(g(\delta))$ is identical to T' since there cannot be any copies of S around (we assumed S not to be in normal form). Set $g(\delta+1)$ and $g(\delta+2)$ to $g(\delta)+1$ and $g(\delta)+2$ respectively. Our condition, that all stack elements of $t'(g(\delta+j))$ arise from $t(\delta+j)$ by replacing copies of S by copies of $\overline{S}$, is retained (for $j=1,2$).

Case 2. So let T' not be in normal form. Let the leftmost redex be $P={}^{\beta}(\lambda x.Q,R)$.

Case 2.1 P lies to the left of every S_i. Let γ be the length of the normal form derivation and set $g(\delta+j)=g(\delta)+j$ for $j<\delta$. Clearly, our condition holds for all these $\delta+i$. The residuals of S in the topmost element of the $t(\delta+i)$ are the same as those of T'.

Case 2.2. There are some S_j, for $j\in J\subseteq k$, that are subterms of P. Each of those is a subterm either of Q or of R. Let J_Q and J_R be the subsets of J containing the indices of subterms of Q and R respectively. We set $g(\delta+i)=g(\delta)+i$ for $i=1,2$. At $t(\delta+1)$ the evaluation of ${}^{\beta}(\lambda x.Q,R)$ is prepared, so our condition holds. So consider $t(\delta+2)$, i.e., the relevant next step in the normal form development of T. First consider those S_q with $q\in J_Q$. Since we assumed that all free variables of S are unbound in T and we took precautions to not accidentally bind variables, we know that S does not contain the variable x. Therefore, the topmost stack element $Q\frac{R}{x}$ contains these S_q as subterms. The replacement does not depend on the structure of S_q, so $t'(g(\delta+2))$ contains $\overline{S}$ at exactly the same positions as $t(\delta+2)$ contains the S_q as subterms. Let us turn to the S_r for $r\in J_R$. In $t(\delta+2)$'s topmost element $Q\frac{R}{x}$, R has been substituted for all occurrences of x in Q and with it all the S_r. Since this substitution is also independent from the syntactic structure of the S_r, again the topmost element of $t'(g(\delta+2))$ can be obtained from replacing all copies of all the S_r by $\overline{S}$.

Case 2.3 There is some $i<k$ such that P is S_i or a subterm of S_i. Let γ be the length of the normal form derivation of S. We set $g(\delta+\gamma)$ to $g(\delta)$. Observe that, by the definition of the normal form derivation, the steps of t between δ and $\delta+\gamma$ are used to determine the normal form of S_i and replace S_i in T' by $\overline{S}$. So the stacks $t(\gamma)$ and $t(\gamma+\delta)$ are of the same height and identical on all but the top layer which differs by the replacement of S_i by $\overline{S}$. Then, obviously our condition holds for $t(\delta+\gamma)$ and $t'(g(\delta))$.

We still need to define g at limits. Suppose κ is a limit and let g be defined on an unbounded subset $D\subseteq\kappa$. Assume without loss of generality, that $\kappa\setminus D$ is unbounded in κ, i.e., cofinally many steps from t are left out in $t\restriction D$. All 'gaps' in D stem from Case 2.3. If there is a stack height σ that appears cofinally often in $t\restriction D$, we know that σ also appears cofinally as stack height in $t\restriction\kappa$. Since the gaps according to Case 2.3 each start and end with the same stack heights and only increase stack height in between, they cannot factor in the $\liminf$ of $t\restriction\kappa$. So we are safe to set $g(\kappa)=\lim_{\gamma<\kappa}g(\gamma)$. Now suppose the stack heights increase unboundedly in $t\restriction D$. Since the gaps only make the stack higher, also the stack heights of $t\restriction\kappa$ increase unboundedly. But in fact, the stack height at the beginning and the end of each gap is the same. Every gap ends before

κ (since D is unbounded), so the part of the stack that gets built up and torn down within a gap cannot factor in to the 'limit stack' (as per Definition 2.10, (c), (ii)). Hence, the first skeleton on the limit stack that appears cofinally often in $t \upharpoonright D$ is the same as in $t \upharpoonright \kappa$. So we are safe to set $g(\kappa) = \lim_{\gamma<\kappa} g(\gamma)$. In both cases the condition holds and residuals are inherited from below the limit. $\square$

2.6 Ordinal λ-definable functions and ordinal computability

We shall now explore which functions on the ordinals are ordinal λ-definable. In his Diploma thesis [Fis10], Tim Fischbach showed how various existing notions of ordinal computability coincide in strength. We tie in our proposed model of ordinal λ-definability into this framework to state our main result at the end of this section.

2.6.1 Primitive recursive set and ordinal functions

A generalization of primitive recursive functions on natural numbers, operating on the universe of sets, has been used in the study of the constructible hierarchy [JK71, Dev73]. In [JK71], Jensen and Karp gave a definition for Prim_O, the class of primitive recursive functions mapping ordinals to ordinals, which is compatible with their notion Prim of primitive recursiveness of functions mapping sets to sets defined alongside in their paper:

Definition 2.18 ([JK71])**.** Let $b_0, \ldots, b_{n-1}$ be unary ordinal functions, i.e., $b_i : \text{Ord} \to \text{Ord}$ for $0 \le i < n$. The symbol $\text{Prim}_O(b_0, \ldots, b_{k-1})$ (*primitive recursive ordinal functions in* $b_0, \ldots, b_{k-1}$) denotes the collection of all functions of type (1) to (5) closed under the schemes for substitution (a) and (b) and recursion (R).

(1) $f(\xi) = b_i(\xi)$ for $0 \le i < k$

(2) $\text{pr}_{n,i}(\vec{\xi}) = \xi_i$, for all $n \in \omega$, $\vec{\xi} = (\xi_1, \ldots, \xi_n)$ and $1 \le i < n$.

(3) $f(\xi) = 0$

(4) $f(\xi) = \xi + 1$

(5) $c(\xi, \zeta, \gamma, \delta) = \begin{cases} \xi, \text{ if } \gamma < \delta \\ \zeta, \text{ else} \end{cases}$

(a) $f(\vec{\xi}, \vec{\zeta}) = g(\vec{\xi}, h(\vec{\xi}), \vec{\zeta})$

(b) $f(\vec{\xi}, \vec{\zeta}) = g(h(\vec{\xi}), \vec{\zeta})$

(R) $f(\xi, \vec{\zeta}) = g(\sup_{\eta<\xi} f(\eta, \vec{\zeta}), \xi, \vec{\zeta})$

We write Prim_O for $\text{Prim}_O()$.

Within this paper, we are interested in the case where the b_i are constant ordinal functions with value α_i. We therefore will write $\text{Prim}_O(\alpha_0, \ldots, \alpha_{k-1})$ in these cases.

A relation on ordinals is Prim_O if its characteristic function is Prim_O.

Let G_1, G_2 denote the inverses of the Gödel pairing function, mapping an ordinal α to the first or second coordinate of the α-th Gödel pair. Based on [JK71, Theorem 4.4] one readily proves:

Lemma 2.19. *Let α be admissible. There is a* Prim_O *relation T such that for any partial $\mathbf{\Sigma}_1(L_\alpha)$-definable function $F : \alpha \rightharpoonup \alpha$ there is a bounded formula ϕ and an ordinal number β such that for all $\gamma \in \alpha$ we have $F(\gamma) = G_1 \min_{\xi\in\alpha} T(\xi, \ulcorner\phi\urcorner, \gamma, \beta)$.*

Proof. Let F be $\mathbf{\Sigma_1}$ via $\exists z\phi$ and the parameter w, i.e.,

$$L(\alpha) \models F(x) = y \leftrightarrow \exists z\phi(x, y, z, w).$$

Assume $x \in \operatorname{dom} F$. Then $F(x) = G_1 \min_{\xi\in\alpha} \phi(x, G_1(\xi), G_2(\xi), w)$. Use the Prim enumeration N of all constructible sets from [JK71] to find a β such that $F(x) = G_1 \min_{\xi\in\alpha} \phi(x, G_1(\xi), G_2(\xi), N(\beta))$. Since ϕ is is Δ_0 and truth of Δ_0 relations is Prim, we get a desired relation T as Prim and via [JK71, 3.5] as Prim_O. $\square$

We obtain a different characterization of primitive recursive functions on ordinals by replacing rules (5) and (R) by:

(5') $e(\xi, \zeta) = \begin{cases} 1 \text{ if } \xi = \zeta \\ 0 \text{ else} \end{cases}$

(R') If g and h are given, define f by

$$\begin{aligned} f(0, \vec{\zeta}) &= g(\vec{\zeta}) \\ f(\xi + 1, \vec{\zeta}) &= h(f(\xi), \xi, \vec{\zeta}) \\ f(\xi, \vec{\zeta}) &= \liminf_{\eta<\xi} f(\eta, \vec{\zeta}) \text{ if } \xi \text{ is a limit ordinal} \end{aligned}$$

It was proved in Tim Fischbach's Diploma thesis [Fis10, Appendix A] that these two schemes are equivalent.

Lemma 2.20 (Fischbach)**.** *The class of* Prim_O *functions and the class obtained from* Prim_O *by replacing (5) by (5') and (R) by (R') are the same.* $\square$

We can now show that the Prim_O functions are $\lambda + E$-definable, the first major step towards our main theorem.

Theorem 2.21 (Seyfferth)**.** *Every* $\mathrm{Prim}_O(\vec{\alpha})$ *function is ordinal $\lambda+E$-definable in $\vec{\alpha}$.*

Proof. A useful device in this proof is the following: Although we cannot forget arguments in the fashion of $\lambda xy.x$ due to the limitations of $\lambda\mathbf{I}$-calculus, we can do so if the argument is an ordinal. With $\mathbf{J} = \lambda z.(((S_c^+, z), \mathbf{I}), \mathbf{I})$ the term $(\mathbf{J}, \underline{\alpha})$ has normal form $\mathbf{I}$ for any α. Define $\mathbf{T}_J = \lambda xy.((\mathbf{J}, y), x)$ and $\mathbf{F}_J = \lambda xy.((\mathbf{J}, x), y)$.

(1) If $\vec{\alpha} = (\alpha_0, \ldots, \alpha_{k-1})$, then every constant function with value α_i is ordinal λ-definable in α_i as $\lambda x.((\mathbf{J}, x), \underline{\alpha_i})$.

(2) To retrieve the i-th input ordinal, consider the following term:

$$P_i = \lambda x_0 x_1 \ldots x_{k-1}.((\ldots(\mathbf{J}, x_0)), (\mathbf{J}, x_1)), \ldots), (\mathbf{J}, x_{k-1})), x_i)$$

The term $(\ldots(P_i, \underline{\alpha_0}), \underline{\alpha_1}), \ldots), \underline{\alpha_{k-1}})$ has normal form $\underline{\alpha_i}$.

(3) As above, with

$$Z = \lambda x_0 x_1 \ldots x_{k-1}.((\ldots(\mathbf{J}, x_0)), (\mathbf{J}, x_1)), \ldots), (\mathbf{J}, x_{k-1})), \underline{0})$$

the term $(\ldots(Z, \underline{\alpha_0}), \underline{\alpha_1}), \ldots), \underline{\alpha_{k-1}})$ has $\underline{0}$ as normal form.

(4) From Example 2.13 it is evident that S_c^+ indeed defines the successor function on ordinals.

(5') The term $\lambda xy.((((((E, x), y), \mathbf{T}_J), \mathbf{F}_J), \underline{0}), \underline{1})$ defines the desired function e.

(a) Let g be defined by a term G and h by H. Define the term $(((G, \underline{\xi}), (H, \underline{\xi})), \underline{\zeta})$. Thanks to our Weak Church-Rosser Theorem, this term's normal form is $\simeq_v$-equivalent to the normal form of $(((G, \underline{\xi}), \underline{h(\xi)}), \underline{\zeta})$ and therefore the term defines the composition function $f(\xi, \zeta) = g(\xi, h(\xi), \zeta)$. The term is straightforward to adapt for more than one parameter ζ. Similarly: (b).

(R') First, let us define the ordered pair of two ordinals α and β as the term $[\underline{\alpha}, \underline{\beta}] = \lambda y.((y, \underline{\alpha}), \underline{\beta})$. Then we have $\overline{([\underline{\alpha}, \underline{\beta}], \mathbf{T}_J)} = \underline{\alpha}$ and $\overline{([\underline{\alpha}, \underline{\beta}], \mathbf{F}_J)} = \underline{\beta}$. Let g be defined by G and h defined by H. Define

$$F_\xi = ((\underline{\xi}, \lambda x. \left[(((H, (x, \mathbf{T}_J)), (x, \mathbf{F}_J)), \underline{\zeta}), (S_c^+, (x, \mathbf{F}_J))\right]), \left[(G, \underline{\zeta}), \underline{0}\right]),$$

i.e., the ξ-fold application of some term $H^* = \left[(((H, (x, \mathbf{T}_J)), (x, \mathbf{F}_J)), \underline{\zeta}),\right.$ $\left.(x, \mathbf{F}_J)\right]$ to the term $G^* = [(G, \underline{\zeta}), \underline{0}]$. Inductively, all approximations ${}^\eta(H^*, G^*)$ for $0 < \eta < \xi$ have as normal form an ordered pair of two ordinals namely $[\underline{f(\eta, \zeta)}, \underline{\eta}]$, and the construction with $\mathbf{T}_J$ and $\mathbf{F}_J$ works at every approximation step. At limits, the pointwise lim inf ensures that limits are taken in both coordinates of the ordered pair. So $\overline{(F_\xi, \mathbf{T}_J)} = \underline{f(\xi, \zeta)}$ and, by parametrizing ξ and ζ, we can easily give a term F such that $\overline{((F, \xi), \zeta)} = \underline{f(\xi, \zeta)}$. The terms are straightforward to adapt for more than one parameter ζ.

This works fine for every $\xi > 0$. For the case of $\xi = 0$ we need to modify F_ξ to include a test for zero: Define

$$F'_\xi = ((((E, \underline{\xi}), \underline{0}), (G, \underline{\zeta})), F''_\xi)$$
$$F''_\xi = ((((((E, \underline{\xi}), \underline{0}), \underline{1}), \underline{\xi}), \lambda x. \left[(((H, (x, \mathbf{T}_J)), (x, \mathbf{F}_J)), \underline{\zeta}), ((S_c^+, x), \mathbf{F}_J)\right]), \left[(G, \underline{\zeta}), \underline{0}\right])$$

The modifications to F''_ξ make sure that its normal form is always an ordinal, even if $\xi = 0$. That way the case distinction in F'_ξ works as intended. □

2.6.2 Minimization

An *ordinal λ-definable predicate* on the ordinals is given by a term P such that $(P, \underline{\alpha})$ takes $\mathbf{T}_I$ as normal form for α in some subset or subclass of the ordinals and $\mathbf{F}_I$ for α in the complement.

In this section, we will see that for every ordinal λ-definable predicate, there is a function defining its least witness. The proof is a generalization of [Bar81, Chapter 9, §2] and Barendregt credits Kleene for the construction. In [Bar81], for any classically λ-definable predicate P on ω, a term H_P is given that has the least witness of the predicate P as normal form. It turns out that the same term yields least witnesses for ordinal λ-definable predicates on Ord under our algorithm. The following definitions are direct adaptations.

Definition 2.22. Following [Bar81, Chapter 9, §2] we define the following terms:

(a) $A_0 = \lambda xwt.((((((((w, \mathbf{T}_I), \mathbf{I}), \mathbf{I}), \mathbf{I}), (t, x)), \mathbf{I}), \mathbf{I}), x)$
(this term is used to escape the recursion)

(b) $A_1 = \lambda xwt.((((w, (t, (S_c^+, x))), (S_c^+, x)), w), t)$
(this term is used to continue the recursion)

(c) $\mathbf{T}_3 = \lambda xy.((((y, \mathbf{I}), \mathbf{I}), \mathbf{I}), x)$
(this term is used to forget an argument of the form A_i)

(d) $\mathbf{F}_3 = \lambda xy.((((x, \mathbf{I}), \mathbf{I}), \mathbf{I}), y)$
(this term is used to forget an argument of the form A_i)

(e) $W = \lambda x.((((x, \mathbf{T}_3), \mathbf{F}_3), A_0), A_1)$
(this term switches between A_0 and A_1 depending on the truth value of x)

The following facts from [Bar81, Chapter 9, §2] hold true for our algorithm. We shall use these as macros in the upcoming proof of Theorem 2.25.

Lemma 2.23 (Seyfferth)**.**

(i) $(\mathbf{T}_I, (\mathbf{T}_3, \mathbf{F}_3)) \simeq_v \mathbf{T}_3$ *and* $(\mathbf{F}_I, (\mathbf{T}_3, \mathbf{F}_3)) \simeq_v \mathbf{F}_3$.

(ii) $\overline{(((A_0, \mathbf{I}), \mathbf{I}), \mathbf{I})} \simeq_v \mathbf{I}$

(iii) $\overline{(((A_1, \mathbf{I}), \mathbf{I}), \mathbf{I})} \simeq_v \mathbf{I}$

(iv) $\overline{((\mathbf{T}_3, A_i), A_j)} \simeq_v A_i$, *for* $i, j \in \{0, 1\}$

(v) $\overline{((\mathbf{F}_3, A_i), A_j)} \simeq_v A_j$, *for* $i, j \in \{0, 1\}$

(vi) $\overline{(W, \mathbf{T}_I)} \simeq_v A_0$

(vii) $\overline{(W, \mathbf{F}_I)} \simeq_v A_1$

Proof. Easily verified by running the algorithm. □

Now we can define the term H_P that has the least witness for P as normal form for any ordinal λ-definable predicate P.

Now we can define the term defining witnesses for predicates.

Definition 2.24. Let P be an ordinal λ-definable predicate. Then define $H_P = \lambda x.((((W, (P, x)), x), W), P)$.

We give the central result of this subsection, the second ingredient to our main result:

Theorem 2.25 (Seyfferth)**.** *Let P be an ordinal λ-definable predicate. Then* $\overline{(H_P, \underline{0})} \simeq_v \underline{\gamma}$ *where* $\gamma = \min_{\gamma \in \mathrm{Ord}}(\overline{(P, \gamma)} \simeq_v \mathbf{T}_I)$ *if such a γ exists. Otherwise* $\overline{(H_P, \underline{0})} \uparrow$.

Proof. We demonstrate that the algorithm works correctly. The computation can be analyzed into the following stages:

- Lines (1) to (8) contain some preliminary setup.
- Lines (9) to (21) form one iteration of the main loop.
- Line (18) marks the time after which the second stack level stabilizes: Through the following iterations, the stack remains stable up to its second level. In fact, at the first limit in time, every finite stack level reached so far contains a term of the same skeleton, cf. line (24).
- Therefore, the same skeleton is assumed at limits and the following successor levels, until eventually level γ is reached in line (26).
- Lines (26) to (42) then reduce the topmost stack level, from a complicated term still containing the subterms used to continue the recursion, to the bare Church numeral $\underline{\gamma}$.
- In the steps abbreviated in line (43), the stack is torn down. Note that at all stack levels, the entire term was pushed onto the next stack level to be evaluated (e.g. lines (18), (24), (26)). Therefore, $\underline{\gamma}$ is handed down directly to the previous stack level without being inserted into some surrounding term. As soon as the first limit level is reached, the stack is immediately pruned to stack height 2 (cf. line (18)) as per Definition 2.10(b)(v).
- In line (44), the stack has height 1 and contains $\underline{\gamma}$ as the desired normal form.

$$(H_P, \underline{0}) \qquad \text{write out } H_P \text{ (1)}$$

$$\overline{(\lambda x.((((W, (P, x)), x), W), P), \underline{0})} \qquad \text{redex (2)}$$

$$1{:}\quad ((((W, (P, \underline{0})), \underline{0}), W), P) \qquad \text{write out } W \text{ (3)}$$

$$(((\overline{(\lambda x.((((x, \mathbf{T}_3), \mathbf{F}_3), A_0), A_1), (P, \underline{0}))}, \underline{0}), W), P) \qquad \text{redex (4)}$$

$$1{:}\quad ((((\overline{(P, \underline{0})}, \mathbf{T}_3), \mathbf{F}_3), A_0), A_1) \qquad \text{WLOG } \overline{(P, \underline{0})} \simeq_v \mathbf{F}_I \text{ (5)}$$

$$((\overline{((\mathbf{F}_I, \mathbf{T}_3), \mathbf{F}_3)}, A_0), A_1) \qquad \text{Lemma 2.23 (6)}$$

$$\overline{((\mathbf{F}_3, A_0), A_1)} \qquad \text{Lemma 2.23 (7)}$$

$$A_1 \qquad \text{return (8)}$$

$$(((A_1, \underline{0}), W), P) \qquad \text{write out } A_1 \text{ (9)}$$

$$((\overline{(\lambda xwt.((((w, (t, (S_c^+, x))), (S_c^+, x)), w), t), \underline{0})}, W), P) \qquad \text{redex (10)}$$

$$1{:}\quad \lambda wt.((((w, (t, \overline{(S_c^+, \underline{0})})), \overline{(S_c^+, \underline{0})}), w), t) \qquad \text{property of } S_c^+ \text{ (11)}$$

$$\lambda wt.((((w,(t,\underline{1})),\underline{1}),w),t) \qquad \text{return} \tag{12}$$

$$(\overline{(\lambda wt.((((w,(t,\underline{1})),\underline{1}),w),t),W)},P) \qquad \text{redex} \tag{13}$$

$$1: \quad \lambda t.((((W,(t,\underline{1})),\underline{1}),W),t) \qquad \text{write out } W \tag{14}$$

$$\lambda t.(((\overline{(\lambda x.((((x,\mathbf{T}_3),\mathbf{F}_3),A_0),A_1),(t,\underline{1}))},\underline{1}),W),t) \qquad \text{redex} \tag{15}$$

$$1: \quad (((((t,\underline{1}),\mathbf{T}_3),\mathbf{F}_3),A_0),A_1) \qquad \text{return} \tag{16}$$

$$\lambda t.(((((((t,\underline{1}),\mathbf{T}_3),\mathbf{F}_3),A_0),A_1),\underline{1}),W),t) \qquad \text{return} \tag{17}$$

$$\overline{(\lambda t.(((((((t,\underline{1}),\mathbf{T}_3),\mathbf{F}_3),A_0),A_1),\underline{1}),W),t),P)} \qquad \text{cofinal skeleton} \tag{18}$$

$$1: \quad (((((((\overline{(P,\underline{1})},\mathbf{T}_3),\mathbf{F}_3),A_0),A_1),\underline{1}),W),P) \qquad \text{WLOG} \tag{19}$$

$$(((((\overline{((\mathbf{F}_I,\mathbf{T}_3),\mathbf{F}_3)},A_0),A_1),\underline{1}),W),P) \qquad \text{Lemma 2.23} \tag{20}$$

$$(((\overline{((\mathbf{F}_3,A_0),A_1)},\underline{1}),W),P) \qquad \text{Lemma 2.23} \tag{21}$$

$$(((A_1,\underline{1}),W),P) \qquad \text{write out } A_1 \tag{22}$$

$$\vdots \qquad \text{similarly} \tag{23}$$

$$\overline{(\lambda t.(((((((t,\underline{2}),\mathbf{T}_3),\mathbf{F}_3),A_0),A_1),\underline{2}),W),t),P)} \qquad \text{cofinal sk.} \tag{24}$$

$$\ddots \qquad \text{eventually (else obvious divergence)} \tag{25}$$

$$\overline{(\lambda t.(((((((t,\underline{\gamma}),\mathbf{T}_3),\mathbf{F}_3),A_0),A_1),\underline{\gamma}),W),t),P)}$$
$$\text{where } \overline{(P,\underline{\gamma})} \simeq_v \mathbf{T}_I \tag{26}$$

$$1: \quad (((((((\overline{(P,\underline{\gamma})},\mathbf{T}_3),\mathbf{F}_3),A_0),A_1),\underline{\gamma}),W),P)$$
$$\text{assumption} \tag{27}$$

$$(((((\overline{((\mathbf{T}_I,\mathbf{T}_3),\mathbf{F}_3)},A_0),A_1),\underline{\gamma}),W),P)$$
$$\text{Lemma 2.23} \tag{28}$$

$$(((\overline{((\mathbf{T}_3,A_0),A_1)},\underline{\gamma}),W),P) \qquad \text{Lemma 2.23} \tag{29}$$

$$(((A_0,\underline{\gamma}),W),P) \qquad \text{write out } A_0 \tag{30}$$

$$((\overline{(\lambda xwt.(((((((w,\mathbf{T}_I),\mathbf{I}),\mathbf{I}),\mathbf{I}),(t,x)),\mathbf{I}),\mathbf{I}),x),\underline{\gamma})},W),P)$$
$$\text{redex} \tag{31}$$

$$1: \quad \lambda wt.(((((((w,\mathbf{T}_I),\mathbf{I}),\mathbf{I}),\mathbf{I}),(t,\underline{\gamma})),\mathbf{I}),\mathbf{I}),\underline{\gamma})$$
$$\text{return} \tag{32}$$

$$(\overline{(\lambda wt.(((((((w,\mathbf{T}_I),\mathbf{I}),\mathbf{I}),\mathbf{I}),(t,\underline{\gamma})),\mathbf{I}),\mathbf{I}),\underline{\gamma}),W)},P)$$
$$\text{redex} \tag{33}$$

$$1: \quad \lambda t.(((((((\overline{(W,\mathbf{T}_I)},\mathbf{I}),\mathbf{I}),\mathbf{I}),(t,\underline{\gamma})),\mathbf{I}),\mathbf{I}),\underline{\gamma})$$
$$\text{Lemma 2.23} \tag{34}$$

$$\lambda t.((((\overline{(((A_0,\mathbf{I}),\mathbf{I}),\mathbf{I})},(t,\underline{\gamma})),\mathbf{I}),\mathbf{I}),\underline{\gamma})$$

$$\text{Lemma 2.23} \quad (35)$$

$$\lambda t.(((\overline{(\mathbf{I}, (t, \underline{\gamma}))}, \mathbf{I}), \mathbf{I}), \underline{\gamma}) \qquad \text{definition } \mathbf{I} \quad (36)$$

$$\lambda t.((((t, \underline{\gamma}), \mathbf{I}), \mathbf{I}), \underline{\gamma}) \qquad \text{return} \quad (37)$$

$$\overline{(\lambda t.((((t, \underline{\gamma}), \mathbf{I}), \mathbf{I}), \underline{\gamma}), P)} \qquad \text{redex} \quad (38)$$

$$1: \quad (((\overline{(P, \underline{\gamma})}, \mathbf{I}), \mathbf{I}), \underline{\gamma}) \qquad \text{assumption} \quad (39)$$

$$(\overline{((\mathbf{T}_I, \mathbf{I}), \mathbf{I})}, \underline{\gamma}) \qquad \text{Lemma 2.23} \quad (40)$$

$$(\mathbf{I}, \underline{\gamma}) \qquad \text{definition } \mathbf{I} \quad (41)$$

$$\underline{\gamma} \qquad \text{return} \quad (42)$$

$$\iddots \qquad \text{note that when going through the stack} \ldots$$

$$\iddots \qquad \ldots \text{also over limits in stack height} \ldots$$

$$\iddots \qquad \ldots \text{no new redexes appear} \quad (43)$$

$$\underline{\gamma} \qquad (44)$$

□

2.6.3 Main result

Theorem 2.26 (Seyfferth)**.** *A partial function $F : \mathrm{Ord} \rightharpoonup \mathrm{Ord}$ on the ordinals is $\lambda + E$-definable in finitely many ordinal parameters if and only if it is $\mathbf{\Sigma}_1$-definable over L.*

Proof. By Lemma 2.19, every ordinal $\mathbf{\Sigma}_1(L)$ definable function can be obtained by one minimization over a Prim_O relation in an ordinal parameter (and by evaluating this result by another Prim_O function). Theorem 2.25 shows that minimization over ordinal $\lambda + E$-definable relations is ordinal $\lambda + E$-definable. Since by Theorem 2.21 all Prim_O functions are ordinal $\lambda+E$-definable, it follows that every $\mathbf{\Sigma}_1(L)$ definable function is already ordinal $\lambda+E$-definable. It remains to show that every ordinal $\lambda + E$-definable function is $\mathbf{\Sigma}_1(L)$ definable. We use the established equivalence of $\mathbf{\Sigma}_1(L)$-definable functions and α-computable functions [KS09] and define an OTM program which computes the normal form derivation of any $\lambda + E$-term:

At any point in the normal form derivation, we would like the current stack content to be coded on our Turing tape. Each stack element is composed of a finite number of terms plus some ordinals. Every term can be represented as the pair of its flesh (a tuple of ordinals) and its skeleton (a finite string over a countable alphabet). So every stack element can be coded into a finite sequence of ordinals. This can be coded into our tape via Gödel pairing $\langle \cdot, \cdot \rangle$: Cell number $\langle \langle n, \gamma \rangle, \xi \rangle = 1$ if and only if the n-th ordinal of stack element γ is greater than ξ. All ordinals coded in this way are preserved as lim inf's across limits, due to the properties of OTMs. The operations on the stack that happen at successor times are all of syntactical nature on terms and can be carried out by an OTM.

At limits in the normal form derivation, our simulation thereof at first will set all ordinals involved to their lim inf:

First, consider the case where some stack height appears cofinal below the limit. We thus are in the case where some term ${}^{\beta}(M, N)$ is approximated. Then all stack levels up to the lim inf of stack heights will have stabilized and the trivial lim inf's are as desired. On the top stack level, however, the OTM lim inf-rule will produce garbage, as both the non-normalized form and the normal form of the approximations ${}^{\gamma}(M, N)$ appear on the cofinal stack height. But OTMs can recognize limits (by the flag-flashing technique introduced in [HL00]) and we can keep track of the stack height on some separate space on the tape (imagine, for simplicity, an extra tape for this task). Whenever a limit is reached, we can check whether there is a cofinally assumed stack height: Simply simulate the computation up to that point time and again, trying out all values from 0 to the number of the current iteration, again flashing a flag whenever the stack has the suspected height to see whether it appears cofinally. This means a vast increase of running time over the length of the normal form derivation to be simulated, but really poses no problem since, in any case, we will be done eventually; nevertheless, time improvements are possible via diagonal enumeration. As soon as the cofinally assumed stack height is known, the desired lim inf's can be found by one additional simulation up to the current step.

In the second case, where no stack height appears cofinally, we are interested in the first (with respect to stack height) skeleton that appears cofinally often in the ever increasing stack. We can search through the limit stack, checking for every possible skeleton whether it appears cofinally and, among those that do, choose the one that appears on the stack first. Once we identified this skeleton, we can search through the limit stack again to determine the desired lim inf. □

We can restrict both the length of the normal form derivation and the stack height in definition 2.10 to some admissible ordinal α: If either reaches α, we say that the normal form derivation diverges. With the resulting notion of *α-normal form derivation* we can define the α-$\lambda + E$-definable (partial) functions on α (possibly in parameters $< \alpha$).

Corollary 2.27. *Let α be admissible. A function partial $F : \alpha \rightharpoonup \alpha$ is α-$\lambda+E$-definable in a finite set of parameters $< \alpha$ if and only if it is $\mathbf{\Sigma}_1$-definable over L_α.*

Proof. As above, Lemma 2.19 and Theorems 2.21 and 2.25 ensure that every $\mathbf{\Sigma}_1(L_\alpha)$-definable function is α-λ-definable in parameters $< \alpha$. For the converse, the admissibility of α ensures that the above construction can be carried out on an α-Turing machine (as definedj, e.g., in [KS09]). □

2.7 Open questions

2.7.1 Strong Church-Rosser Theorem

We already stated Conjecture 2.16, that a strong version of the Church Rosser Theorem holds for our calculus.

Conjecture. *Let T be a term with normal form and let $S \subseteq T$ be a subterm. Then $\overline{T} \simeq_v \overline{T\left[\frac{\overline{S}}{S}\right]}$.*

2.7.2 $\lambda + Z$-definability

Another open problem is whether we can prove our main result for calculi seemingly weaker than $\lambda + E$. We added a predicate for equality of ordinals to our calculus and defined the $\lambda + E$-definable functions to obtain the initial function (5') in the definition of primitive recursive functions. All other arguments from Section 2.6 go through also for λ-definable functions. We conjecture that we can replace the predicate E by a predicate Z that tests numerals for being zero, in the same fashion as E tests for equality.

Conjecture. *Theorem 2.26 holds for $\lambda + Z$-definable functions.*

Proof idea. Equality of ordinals can be defined recursively from a test for 0:

$$\begin{aligned}&\alpha = \beta \leftrightarrow (\alpha = 0 \wedge \beta = 0) \vee (\alpha \leq \beta \wedge \beta \leq \alpha)\\ &\alpha \leq \beta \leftrightarrow \forall \gamma \in \alpha\, \exists \delta \in \beta\, \gamma = \delta\end{aligned}$$

One should be able to carry out this recursion in our ordinal λ-calculus. The $\exists$ and $\forall$ quantifiers can be modeled in the following way: Let P be a predicate on the ordinals and assume we want to decide, e.g., whether P holds for some $\gamma < \alpha$. We define a function $f : \mathrm{Ord} \mapsto \{0, 1\}$ such that $f(\alpha) = 1 \leftrightarrow \exists \gamma < \alpha\, P(\alpha)$.

$$\begin{aligned}f(0) &= 0\\ f(\gamma + 1) = 1 &\leftrightarrow P(\gamma) \vee f(\gamma) = 1\\ f(\mu) &= \liminf_{\nu<\mu} f(\nu) && \text{if } \mu \text{ is a limit}\end{aligned}$$

This primitive recursion is $\lambda + Z$-definable if P is so.

Due to Note 2.9, it seems unlikely that we are able to go even weaker than $\lambda + Z$. Since, syntactically, the numeral for 0 is indistinguishable from the numerals for non-zero ordinals, it appears doubtful to obtain a test for zero by syntactical tricks. Also, its arithmetical properties cannot be validated without a means to talk about equality of ordinals.

2.7.3 Variations of the model

As with the other models of ordinal computation, there are interesting variations imaginable. While λ-calculus does not come with a canonical distinction between time and space, our normal form algorithm can easily be restricted in both runtime or stack height. Asymmetric models, such as ITTMs or the restriction of OTMs in Part II of this thesis, have interesting theories so these two paths, i.e., restricting stack height but not runtime and restricting input complexity but neither runtime nor stack height, should be explored. In an early stage of the development, the author conjectured that the present calculus restricted to finite stacks (and consequently without part (c)(ii) of Definition 2.10) would be equivalent in strength to the Prim_O functions. While plausible from interpreting terms of the form ${}^{\alpha}(M, N)$ as some kind of `for`-loops, it turned out that this is false: Due to its un-typed nature, the thus defined generalization of λ-calculus is capable of giving 'primitive recursive' definitions for functionals such as the Ackermann function, while the calculus of primitive recursive functions is limited to defining only functions on ordinals in a recursive manner.

Moving away from looking at the calculus solely as means of defining functions on the ordinals, the work done to generalize λ-calculus may perhaps be used to extend other calculi that are centered on the re-writing of terms to the transfinite. The author hopes for the present work to be helpful in further studies in this direction.

Part II

Descriptive set theory of ordinal Turing machines

Chapter 3

Introduction to descriptive set theory and ordinal Turing machines

3.1 Introduction

The study of subsets of real numbers is a classical topic in mathematics. Today, descriptive set theory is one of the major active fields in set theory, using techniques from constructibility, infinite games, and forcing to analyze the properties of complexity classes of sets of reals. The complexity of such sets is usually measured by how simple a definition (written in the language of second order arithmetic) for the set can be given. The following brief overview is based on Akihiro Kanamori's excellent introduction in [Kan03].

The language of second order arithmetic refers to statements and terms over $\{\mathrm{ap}, +, \times, \exp, <, 0, 1\}$ using variables of type 0 $\{v_i^0 \mid i \in \omega\}$ and type 1 $\{v_i^1 \mid i \in \omega\}$ and where ap takes an argument of type 0 and 1 each and has values in type 0 ($+$, $\times$, exp, $<$, 0, 1 are, as suggested by their symbols, functions, a relation and constants on type 0). We use the connectives $\{\neg, \vee, \wedge, \rightarrow, \leftrightarrow\}$ and the so called number quantifiers $\{\exists^0, \forall^0\}$ and function quantifiers $\{\exists^1, \forall^1\}$ (ranging over their respective types of variables) to build formulas. Informally, we use variables k, l, m, etc. for variables of type 0 and x, y, z for type 1 and omit the type of the quantifiers where possible. The formulas are evaluated over the structure $({}^\omega\omega, \omega, \mathrm{ap}, +, \times, \exp, <, 0, 1)$. The function ap interprets elements of ${}^\omega\omega$ as functions on ω and takes as arguments an $x \in {}^\omega\omega$ and a $k \in \omega$ and returns the value $x(k) \in \omega$. Bounded quantification refers to formulas of the form $\exists n(n < m \wedge \phi)$ and $\forall n(n < m \rightarrow \phi)$ and is abbreviated as $\exists n < m\ \phi$ and $\forall n < m\ \phi$ respectively.

For our considerations, two of the various hierarchies employed in descriptive set theory are relevant. First consider the *lightface arithmetical hierarchy* on these formulas:

A formula is Δ_0^0 if only bounded quantification is used.
A formula is Σ_1^0 if it is of the form $\exists k\phi$ where ϕ is Δ_0^0.
A formula is Π_1^0 if it is of the form $\forall k\phi$ where ϕ is Δ_0^0.
A formula is Σ_{n+1}^0 if it is of the form $\exists k\phi$ where ϕ is Π_n^0.
A formula is Π_{n+1}^0 if it is of the form $\forall k\phi$ where ϕ is Σ_n^0.

A set of reals $A \subseteq {}^\omega\omega$ is Σ_n^0 (or Π_n^0) if it has a Σ_n^0 (or Π_n^0) definition ϕ, i.e., $x \in A \leftrightarrow \phi(x)$. The set A is called Δ_n^0 if it is both Σ_n^0 and Π_n^0. It is worth noting that the Σ_1^0 sets are precisely the classically enumerable sets and consequently the computable sets of reals coincide with the class Δ_1^0.

Next, we introduce the *lightface analytical hierarchy*:

A formula is Σ_1^1 if it is of the form $\exists x\phi$ where ϕ is arithmetical.
A formula is Π_1^1 if it is of the form $\forall x\phi$ where ϕ is arithmetical.
A formula is Σ_{n+1}^1 if it is of the form $\exists x\phi$ where ϕ is Π_n^1.
A formula is Π_{n+1}^1 if it is of the form $\forall x\phi$ where ϕ is Σ_n^1.

We will also set both Σ_0^1 and Π_0^1 to Σ_1^0.

Again, a set $A \subseteq {}^\omega\omega$ is Σ^1_n (or Π^1_n or Δ^1_n) if it has a Σ^1_n (or Π^1_n or Δ^1_n) definition ϕ. The set A is called Δ^1_n if it is both Σ^1_n and Π^1_n.

In fact, analytical sets also have a bit more streamlined definitions, which will shall use below:

A set A is Σ^1_1 if $x \in A \leftrightarrow \exists y \forall k \phi(x \restriction k, y \restriction k, k)$ where ϕ is Δ^0_0.
A set A is Π^1_1 if $x \in A \leftrightarrow \forall y \exists k \phi(x \restriction k, y \restriction k, k)$ where ϕ is Δ^0_0.

3.2 Tree representations

We consider the following type of trees:

Definition 3.1.

(a) A tree on X is a subset of ${}^{<\omega}X$ that is closed under initial segments. An infinite branch corresponds to an element of ${}^\omega X$.

(b) A tree on $X \times Y$ is a subset of ${}^{<\omega}X \times {}^{<\omega}Y$ of same length sequences, closed under pointwise initial segments.

(c) For a tree T on $X \times Y$ and some $x \in {}^\omega X$ or $x \in {}^{<\omega}X$, we denote by T_x the tree on Y of those sequences y that are compatible with x in T, i.e., that for all $n \in \omega$ we have $(x \restriction n, y \restriction n) \in T$.

(d) Trees without infinite branches are called well-founded.

The definition for trees on $X \times Y$ has an obvious generalization to finite tuples $X_0 \times X_1 \times \ldots X_{k-1}$. We shall in the following only consider trees where the X_i are ordinals.

Fact 3.2. *When ordered by reverse coordinate-wise inclusion, a tree on $\alpha_0 \times \alpha_1 \times \ldots \alpha_{k-1}$ is well-founded if it has a strictly order-preserving map into the ordinals (we will call such a map an* order-preserving embedding*).*

Π^1_1 sets have the following tree representation, which we shall call the *Luzin-Sierpiński tree*: $A \subseteq {}^k({}^\omega\omega)$ is Π^1_1 if and only if there is a recursive tree T on ${}^k\omega \times \omega$ such that

$$\begin{aligned} x \in A \leftrightarrow & T_x \text{ is well-founded} \\ & \leftrightarrow \text{there is an order-preserving embedding} \\ & \quad \text{of } T_x \text{ into some countable ordinal} \end{aligned}$$

For a proof we refer the reader to textbooks on descriptive set theory, one example being [Kan03, Theorem 13.1]. The general idea is: given the Π^1_1 definition $x \in A \leftrightarrow \forall y \exists k \phi(x \restriction k, y \restriction k, k)$ for some Δ^0_0 ϕ, define T to be the tree of all pairs of length k sequences (u, v) so that $\neg\phi(u \restriction n, v \restriction n, n)$ for all $n \leq k$. There, infinite branches occur only in subtrees induced by reals in ${}^\omega\omega \setminus A$.

The Shoenfield tree is a tree representation for Σ^1_2 sets: For a Σ^1_2 set $B \subset {}^k({}^\omega\omega)$ let A be Π^1_1 such that $B = \{x \mid \exists y (x, y) \in A\}$. Let T be the Luzin-Sierpiński tree for A. The Shoenfield tree S for B is such that

$$x \in B \leftrightarrow S_x \text{ has an infinite branch}$$

where an infinite branch of S_x codes

- a real y such that $(x, y) \in A$ and
- some order-preserving embedding of $T_{(x,y)}$ into some countable ordinal α

This can be coded into an element of ${}^{\omega}\omega_1$, therefore S is a tree on ${}^{k}\omega \times \omega_1$. S_x is a tree on ω_1.

Since the Shoenfield tree can be constructed within L and even in every model of KP that contains ω_1 as a subset, we can state the following fact:

Fact 3.3 (Shoenfield absoluteness). *Every Σ^1_2 relation is absolute for transitive models of* KP *that contain ω_1 as a subset.*

This will be re-proved from the perspective of ordinal computability, along with several further facts from classical descriptive set theory:

Fact 3.4 (Σ^1_2 uniformization). *For every Σ^1_2 relation $R \subseteq {}^{\omega}\omega \times {}^{\omega}\omega$ there is a Σ^1_2 relation $U \subseteq R$ such that for every x with $\exists y(x, y) \in R$ there is precisely one z with $(x, z) \in U$.*

A *norm* on some set A is a map $\phi : A \to \mathrm{Ord}$. A *pre-wellordering* is a relation that is total, transitive and well-founded, but not necessarily anti-symmetric. A Σ^1_2 *norm* is a norm for which there are a Σ^1_2 pre-well-ordering P and a Π^1_2 pre-well-ordering Q such that:

$$x \in A \wedge \phi(x) \leq \phi(y) \leftrightarrow P(x, y) \leftrightarrow Q(x, y)$$

Fact 3.5 (Σ^1_2 norms). *Every Σ^1_2 set has a Σ^1_2 norm.*

When allowing a real parameter in defining the above hierarchies, we get boldface versions of the defined pointclasses. The following is often stated in its boldface version as Suslin's theorem (every $\boldsymbol{\Sigma}^1_2$ set is union of ω_1-many Borel sets), but this lightface version also holds true.

Fact 3.6 (Suslin). *Every Σ^1_2 set is the union of ω_1-many Δ^1_1 sets.*

3.3 Ordinal Turing machines

Using transfinite recursion, we can generalize Turing computations to the infinite. At successor times, the program P is used as in the standard Turing machine case, with the single exception that when dealing with tapes of transfinite length, a convention has to be found what should happen when a read-write-head is being moved left from a cell indexed by a limit ordinal. In this situation we want the head to be reset to the beginning of the tape (the leftmost cell).

For limit times, the Turing-program cannot determine the tape content, head positions, and program state, so we have to define them in a sensible way. Following the lines of [Koe05] we use inferior limits: We want each single cell of every tape to contain the lim inf of its previous values, the machine state to be the lim inf of the previous machine states and every tape's read/write-head to be located on the cell indexed by the lim inf over the positions it previously assumed in the limit machine state, i.e., the least cell that was read cofinally often while the machine was in the same state as at the limit time. The lim inf of the program states may be imagined as an instruction starting a loop that is carried out cofinally often before the limit time. So setting the head according to that loop ensures that the loop instruction when called *at* the limit time operates on the same part of the tape it operated on cofinally often before.

More formally:

Definition 3.7 ([KS09])**.** Let α be a limit ordinal or $\alpha = Ord$. Let $P \subseteq \{0,1\} \times \omega \times \{0,1\} \times \omega \times \{-1,1\}$ be finite and let $T_0 \in ({}^{\alpha}\{0,1\})$ be the initial tape content of the tape of length α. A triple

$$(T_\theta, H_\theta, S_\theta)_{\theta \leq \Theta}$$

is called an *α-Turing computation by P on input* T_0 if the following conditions hold:

(a) $\Theta \leq \alpha$;

(b) $S_\theta \in \omega$ for $\theta \leq \Theta$;

(c) $H_\theta \in \alpha$ for $\theta \leq \Theta$;

(d) $T_\theta : \alpha \to \{0,1\}$ for and for $\theta \leq \Theta$;

(e) $(T_\theta, H_\theta, S_\theta)_{\theta \leq \Theta}$ is defined recursively in P and the initial tape contents T_0 in the following way:
Termination: Let $\theta \leq \Theta < \alpha$ and let $(T_{\theta'}, H_{\theta'}, S_{\theta'})_{\theta' \leq \theta}$ be already defined. If there is no $(a, s, a', s', d) \in P$ where $T_\theta(H_\theta) = a$ and $S_\theta = s$ then the computation *halts* or *terminates*, i.e., $\theta = \Theta$.
Successor step: Let $\theta < \Theta$, let $(T_{\theta'}, H_{\theta'}, S_{\theta'})_{\theta' \leq \theta}$ be already defined and let there be a $c = (a, s, a', s', d) \in P$ where $T_\theta(H_\theta) = a$ and $S_\theta = s$. Choose c minimally with respect to some fixed well-order on P. As usual we want the configuration $(T_{\theta+1}, H_{\theta+1}, S_{\theta+1})$ to be derived from $(T_\theta, H_\theta, S_\theta)$ according to the instruction c.
We require:

$$T_{\theta+1}(\xi) = \begin{cases} a & \text{if } \xi = H_\theta \\ T_\theta(\xi) & \text{otherwise} \end{cases}$$

$$H_{\theta+1} = \begin{cases} H_\theta + 1 & \text{if } d = +1 \\ H_\theta - 1 & \text{if } d = -1 \text{ and } H_\theta \text{ is a successor ordinal} \\ 0 & \text{if } d = -1 \text{ and } H_\theta \text{ is a limit ordinal} \end{cases}$$

$$S_{\theta+1} = s'.$$

Limit step: Now let $\theta \leq \Theta$ be a limit ordinal and let $(T_{\theta'}, H_{\theta'}, S_{\theta'})_{\theta' < \theta}$ be already defined. For $\xi < \alpha$ set

$$S_\theta = \liminf_{\theta' < \theta} S_{\theta'}$$

$$H_\theta = \liminf_{\theta' < \theta, S_\theta = S_{\theta'}} H_{\theta'}$$

$$T_\theta(\xi) = \liminf_{\theta' < \theta} T_{\theta'}(\xi).$$

Divergence: Note that the machine configuration at limit times is always defined whenever the configurations at all previous stages are defined. If $\theta = \Theta = \alpha$ we say that the computation *diverges*.

In case of $\alpha = \infty = \mathrm{Ord}$ we speak of an *ordinal* Turing computation (*OTM-computation*).

It is described in depth in [Koe05] how these machines can be used to compute a bounded truth predicate for the constructible hierarchy L by

(a) enumerating codes for all elements in L and

(b) enumerating bounded formulas (i.e., formulas in which all quantifiers appear bounded in the sense of $\exists x \in y$).

These enumerations have to be carefully arranged in a way that for a given pair of bounded formulas and an assignment of elements from L, truth can be checked by looking up truth values of pairs of formulas and assignments determined previously. The result enables us to effectively search L for witnesses of any given Σ_1 formula of set theory.

There are various forms of initial tape contents that can be interpreted as inputs to computations. For the following we are interested in OTM-computable sets of reals.

Definition 3.8. A set $A \subseteq {}^{\omega}2$ is called *OTM-enumerable* if there is a Turing program P such that for every $x \in {}^{\omega}2$ the OTM-computation by P on initial tape content x (written on the first ω-many cells, all other cells being filled with 0s) halts if and only if $x \in A$.

The set A is called *OTM-decidable* if both A and ${}^{\omega}2 \backslash A$ are OTM-enumerable.

We give here a slight improvement over [Koe05, Lemma 2.6].

Lemma 3.9. *Let M be a transitive model of* KP*, i.e., of Kripke-Platek set theory. Let P be a program and let $T(0) : \mathrm{Ord} \to 2$ be an initial tape content so that $T(0) \restriction (\mathrm{Ord} \cap M)$ is Δ_1 definable in M. Let $S : \theta \to \omega$, $H : \theta \to \mathrm{Ord}$, $T : \theta \to {}^{\mathrm{Ord}}2$ be the ordinal computation by P with input $T(0)$. Then:*

(a) The ordinal computation by P with input $T(0)$ is absolute for M below $(\mathrm{Ord} \restriction M)$, i.e., $S : \theta \to \omega \cap M$, $H : \theta \to \mathrm{Ord} \cap M$, $\bar{T} : \theta \to {}^{\theta \cap M}2$ with $\bar{T}(t) = T(t) \restriction (\mathrm{Ord} \cap M)$ is the ordinal computation by P with input $T(0) \restriction (\mathrm{Ord} \cap M)$ as computed in the model M.

(b) If $\mathrm{Ord} \subseteq M$ then the ordinal computations by P in M and in the universe V are equal.

(c) Let $\mathrm{Ord} \subseteq M$ and $x, y \subseteq \mathrm{Ord}$, $x, y \in M$. Then $P : \chi_x \mapsto \chi_y$ if and only if $M \models P : \chi_x \mapsto \chi_y$.

(d) Let $x, y \subseteq \mathrm{Ord}$, $x, y \in M$. Assume that $M \models P : \chi_x \mapsto \chi_y$. Then $P : \chi_x \mapsto \chi_y$.

Proof. We introduce the requirement that the initial tape content $T(0)$ be Δ_1 definable in M in order for the recursive definition of a computation to become a Σ_1 recursion. So the recursion as described in [Koe05] can be carried out in KP. □

3.4 Computing Δ^1_2

Theorem 3.10. *A set A of reals is OTM-enumerable if and only if it is Σ^1_2.*

Proof. Let A be OTM-enumerable, i.e., there is a Turing program P such that $a \in A$ if and only if P halts on input a. Now the latter condition is Σ^1_2: *There is* a real coding a halting computation on input a; to express coding a computation of an ordinal machine requires to check the wellfoundedness of the 'time-axis'; this can be done by another *for all* quantifier, as well-foundedness is a Π^1_1 property.

Conversely, if A is Σ^1_2 by the formula ϕ, then by the Shoenfield absoluteness theorem about the absoluteness of Σ^1_2 properties we get: $a \in A$ if and only if $L[a] \models \phi(a)$ [Jec03, Theorem 25.20]. In models of set theory, ϕ is uniformly equivalent to a Σ_1 formula ψ of set theory [Jec03, Lemma 25.25]. So $a \in A$ if and only if $L[a] \models \psi(a)$. But the latter can be (semi-)computed by an OTM as outlined above and proved in [Koe05]: successively build up the $L[a]$-levels and check whether $\psi(a)$ is true in the level; if yes, then stop, otherwise continue. So A is OTM-enumerable. □

Corollary 3.11. *A set A of reals is OTM-computable if and only if it is Δ^1_2.*

3.5 Summary of the next chapter

The paper had its origin in the characterization of the OTM-decidable sets of reals as Δ^1_2 just stated, which was proved by Koepke and the author. The author then realized that Shoenfield absoluteness, the central tool in this proof, itself can be proved by the means of OTMs: There is an algorithm that finds the infinite branches in the Shoenfield tree defined in the original proof. Philipp Schlicht then suggested some more facts like Σ^1_2 uniformization and Σ^1_2 norms from descriptive set theory that could be proved as corollaries to the algorithmic approach. Together Schlicht and the author defined an alternative tree representation for Σ^1_2 sets, derived from OTM computations, that can play the role of the Shoenfield tree in both the original and the algorithmic proof. The tree-searching algorithm used to search the Shoenfield tree prompted the author to give a definition of nondeterministic OTMs and to prove that they are not stronger than deterministic OTMs when it comes to decide sets of reals. Schlicht suggested stratifying the pointclasses between Π^1_1 and Σ^1_2 via bounds on the halting time of our algorithm and proving uniformization and existence of norms for these classes. The previously defined tree of partial computations was used in proving these results. Finally, Schlicht and the author briefly discuss the Σ^1_2 universality of the OTM's halting problem, and how the complexity and the universality properties of iterated jumps depend on the set theoretic setting.

Chapter 4

Tree representations and ordinal machines [SS12] [1]

[1]in contrast to the published version, the version printed here includes authorship tags for theorems, definitions, etc.

4.1 Introduction

Ordinal computability studies generalized computability theory by means of classical machine models that operate on ordinals instead of natural numbers. Starting with Joel Hamkins' and Andy Lewis' Infinite Time Turing Machines (ITTM) [HL00], recent years have seen several of those models which provided alternate approaches and new aspects for various ideas from logic, set theory and classical areas of generalized computability theory. With ITTMs, the machine may carry out a transfinite ordinal number of steps while writing 0s and 1s on tapes of length ω. This is achieved by the addition of a limit rule that governs the behavior of the machine at limit times. The 0s and 1s on the ω-long tape are interpreted as subsets of ω (*reals*). It turns out that the sets of reals semi-decidable by these machines form a subset of Δ^1_2. Similar studies have been carried out for infinite time register machines (ITRMs), whose computable reals are exactly the reals in $L_{\omega_\omega^{CK}}$ [Koe09].

Another direction of ordinal computability lifts classical computability to study not the subsets of ω, but of an arbitrary ordinal α, or even the class Ord of all ordinals. In this case, both space and time are set to that ordinal α, i.e. in the Turing context, we deal with machines that utilize a tape of length α and either stop in less than α many steps or diverge. The computation is steered by a standard Turing program and a finite number of ordinal parameters less than α so the machines can talk about ordinals below α in much the same fashion as classical Turing machines can about natural numbers. This approach unveils strong connections to Gödels universe of constructible sets and the classical work on α-recursion theory [KS09].

In the present paper, we aim between these two approaches by analyzing the computable sets of reals of Turing machines with Ord space *and* time but without allowing arbitrary ordinal parameters. Omission of the parameters leads to a model in which all computational information is contained in the real input, the finite Turing program and the limit rules. We work with *ordinal Turing machines (OTMs)*, the machine model introduced in [Koe05]. Let us briefly review the basic features, for more detail and background the reader is referred to the original paper.

An OTM uses the binary alphabet on a one-sided infinite tape whose cells are indexed by ordinal numbers. At any ordinal point in time, the machine head is located on one of these cells and the machine is in one of finitely many machine states indexed by natural numbers. Since we utilize both Ord space and time, there is no need to use multiple tapes in our definition; any fixed finite number of tapes can be simulated by interleaving the tapes into one. A typical program instruction has the form $(a, s, a', s', d) \in \{0, 1\} \times \omega \times \{0, 1\} \times \omega \times \{-1, 1\}$ and is interpreted as the instruction *"If the symbol currently read by the machine's read-write head is a and the machine is currently in state s, then overwrite a with the symbol a', change the machine state to s', and move the head according to d either to the left or to the right"*. At successor times in the course of the computation, the machine behaves like a standard Turing machine, with the following exception: If the machine head rests on a cell indexed by a limit ordinal or 0 and a *"move left"*-instruction is carried out, then the head is set to position 0. The machine accesses the transfinite by the following rule, known as the lim inf-rule:

At a limit time λ, the machine state is set to the lim inf of the states of previous time, i.e., the least state that was assumed cofinally often in the previous steps. Similarly, we set the tape content for each cell individually to the lim inf of the previous cell contents; in other words, a cell contains a 0 at time λ if it contained a 0 cofinally often before λ, and it contains a 1 at time λ otherwise. We also set the head position to the cell indexed by the lim inf over the indices of the cells visited at previous steps in which the machine's state matched the limit stage.

These ordinal machines may be used to describe sets of reals. In order to input a real number into an ordinal Turing machine, we start the computation with an initial tape content coding the real; so the initial tape contents is a sequence of the numbers 0 and 1 written in the cells with finite index. Note that our basic definitions do not involve ordinal parameters as in [Koe05, Definition 2.5], hence our main results refer to pointclasses defined without parameters. Since elements of ${}^{\omega}\omega$ can be coded in ${}^{\omega}2$ via Gödel pairing, we can have elements of ${}^{\omega}\omega$ as input as well. Thus we will also refer to elements of ${}^{\omega}\omega$ as real numbers. Let us denote the OTM computation by a program P on input x as $P(x)$ and abbreviate the statement "$P(x)$ halts" as $P(x)\downarrow$. The notion of *input and output* of an OTM computations is defined as in [Koe05]. We will say that a partial function $f : X \rightharpoonup Y$ is *OTM computable* if there is a program P that halts on input $x \in \operatorname{dom} f$ with output $f(x) \in Y$, given a suitable coding of elements of X and Y into OTM tapes.

Definition 4.1 (Koepke)**.** A set of reals $A \subseteq {}^{\omega}\omega$ is called *OTM semi-decidable* if there is an ordinal Turing machine that halts if and only if the initial tape content was an element of A. A is called *OTM decidable* if both A and ${}^{\omega}\omega \setminus A$ are OTM semi-decidable.

Our motivation is to use ordinal machines to refine uniformization results in descriptive set theory. In [Hjo10] it is shown that many results in descriptive set theory have simple proofs using admissible sets; we go further than [Hjo10] in providing explicit algorithms for the constructions. In Section 4.2, we shall define an algorithm for searching for infinite branches in the Shoenfield tree. This implies that the Σ^1_2 sets of reals are exactly the OTM semi-decidable sets of reals. As a consequence, we will re-establish Shoenfield's absoluteness theorem from the perspective of ordinal computability. The fact that the Σ^1_2 sets of reals are exactly the OTM semi-decidable sets of reals may be alternatively obtained from Σ^1_2 absoluteness and the fact that bounded truth in L is an OTM computable relation (for the latter see [Koe05]). In Section 4.3, we shall introduce a tree representation for Σ^1_2 sets that is based on finite fragments of OTM computations. We will apply the algorithm in Section 4.2 to this tree representation to obtain our main result: Uniformization for classes of sets OTM semi-decidable by computations with input-dependent upper bounds on the halting time. Section 4.4 introduces a notion of nondeterministic OTM computations and establishes that nondeterministically OTM decidable sets are already deterministically so. We shall then show that the jump structure of our machines depends on set theoretic assumptions.

4.2 Computing the Shoenfield tree

In this section, we define an OTM algorithm searching for branches in the Shoenfield tree. To assist in the computations, let us fix the following OTM computable functions. The Gödel pairing function is a bijection $\langle\cdot,\cdot\rangle : \mathrm{Ord} \times \mathrm{Ord} \to \mathrm{Ord}$. Elements of Baire space can be represented as subsets of ω by coding their graph via Gödel pairing. The function $o : \omega \to {}^{<\omega}\omega$ is a computable bijection providing a computable enumeration of the basic open sets $O(i)$ of the Baire space ${}^{\omega}\omega$, where $O(i)$ denotes the basic open set defined by the sequence $o(i)$. Let us recall some basic notation commonly used in classical descriptive set theory: If T is a tree on ${}^{k}\omega \times \alpha$ and $x \in {}^{k}({}^{\omega}\omega)$, let $T_x = \{u \in {}^{n}\alpha : (x \restriction n, u) \in T,\ n \in \omega\}$. If $s \in {}^{k}({}^{m}\omega)$, let $T_s = \{u \in {}^{n}\alpha : (s \restriction n, u) \in T,\ n < m\}$.

We will make use of the standard tree representation for Π^1_1 sets due to Luzin and Sierpiński. Recall that a set $B \subseteq {}^{k}({}^{\omega}\omega)$ is Π^1_1 if there is a tree T on ${}^{k}\omega \times \omega$ such that the relation $\{(x,i) \mid o(i) \in T_x\}$ is computable and $x \in B$ if and only if T_x is well-founded. Let us call T the *Luzin-Sierpiński tree* for B. The tree T_x is well-founded if and only if there is an order-preserving embedding of T_x into some countable ordinal α. Hence, to check whether $x \in B$, we can look for a suitable infinite branch in the tree S on ${}^{k}\omega \times \omega_1$ of all pairs (s,u) with $s \in {}^{k}({}^{n}\omega)$ and $u \in {}^{n}\omega_1$ for some $n \in \omega$ where u codes an order-preserving map $f_u : T_s \cap \{o(i) \mid i < \mathrm{length}(u)\} \to \omega_1$. This is the *Shoenfield tree* projecting to B.

Let us first define an algorithm searching the Shoenfield tree for a Π^1_1 set $B \subseteq {}^{\omega}\omega$. Let T be the Luzin-Sierpiński tree for B. We would like the algorithm to halt on input $x \in {}^{\omega}\omega$ if and only if $x \in B$. Depth-first-search (DFS) is employed to find an infinite branch in the subtree of S that consists of the pairs (s,u), where $s = x \restriction n$ for some $n \in \omega$. In other words, we will search S_x, which is a tree on ω_1. Clearly, membership in S of any given pair (s,u) is OTM decidable inside every admissible set, as the property $o(i) \in T_s$ is computable in the classical sense. Note that ω may be used as a constant, since the constant function with value ω is OTM computable.

Algorithm 4.2 (Seyfferth).

```
set $\alpha = 0$

MAIN:
set $u = ()$;
set $n = 0$;
call DFS($u$);
increment $\alpha$;
call MAIN;

DFS($u$):
if $n = \omega$ then stop;
if $(x \restriction n, u) \in S$ then increment $n$ and set $u = u \frown 0$ and
call DFS($u$) and decrement $n$ and set $u = u \restriction n$;
if $u(n) < \alpha$ increment $u(n)$ and call DFS($u$);
```

The algorithm starts with the empty sequence $u = ()$ and in stage $\alpha = 0$. Whenever DFS(u) is called, all possible extensions of u by a single ordinal $\beta < \alpha$ are considered. When all $\beta < \alpha$ have been tried, DFS(u) ends. If a extension

$u \frown \beta \in S_x$ is found, the recursion will immediately try to extend it further and $\mathrm{DFS}(u \frown \beta)$ is called. Whenever the algorithm tries an extension $u \frown \beta$ that is not in S_x, this extension is not followed further and $u \frown (\beta + 1)$ is tried next. If the length n of u has reached ω, a branch is found, i.e., u codes an order preserving embedding of T_x into the ordinal α. If no branch can be found, the recursion eventually breaks down, α is incremented, and the algorithm starts over with the empty sequence.

Throughout the algorithm, the variable u is stored in an extra tape whose n-th cell contains a 1 if and only if $n = \langle p, q\rangle$ and $u(p) \geq q$ and 0 otherwise. Therefore, the variable also contains the desired value at limit times.

Lemma 4.3 (Seyfferth)**.** *The algorithm will find the lexicographically least infinite branch through S_x, if there is one.*

Proof. It is clear that if the algorithm finds a branch, it will find the lexicographically least. So we have to show that this branch is eventually found. Let $v \in {}^{\omega}\omega_1$ be the lexicographically least branch of S_x and let γ be the supremum of the ordinals in v. The tree $S_x \cap {}^{<\omega}\gamma$ is countable. Observe that the algorithm visits exactly the nodes of $S_x \cap {}^{<\omega}\gamma$ in the stages $\alpha < \gamma$ and that every node is visited only once. Since this subtree contains no infinite branches, the algorithm sets $\alpha = \gamma$ after countably many steps. Note that in stage γ, the algorithm will first visit the countably many sequences $w \in S_x$ that are lexicographically smaller than $v \upharpoonright \mathrm{length}(w)$. No node w that is lexicographically greater than $v \upharpoonright \mathrm{length}(w)$ is visited before the algorithm examines every initial segment of v, so the algorithm eventually finds the branch in countable time. □

Now consider a Σ^1_2 set $A \subseteq {}^{\omega}\omega$ and a Π^1_1 set $B \subseteq {}^{\omega}\omega \times {}^{\omega}\omega$ such that $p(B) = A$. We will modify the previous algorithm to semi-decide the set A. Let $T \subseteq ({}^{2}\omega \times \omega)^{<\omega}$ be the Luzin-Sierpiński tree for B. The Shoenfield tree S for B is the tree of all (s, t, u) where u codes an order-preserving embedding $f_u : T_{s,t} \to \mathrm{Ord}$. Since $B = p([S])$ and $A = p(B)$, we have $x \in A$ if and only if the tree $S_{\vec{x}}$ (on $\omega \times \omega_1$) has an infinite branch. In order to find such a branch for a given x, the algorithm proceeds in stages $\alpha \in \mathrm{Ord}$. In each stage α, depth-first-search is employed to find an infinite branch in the subtree of $S_{\vec{x}}$ which consists of the pairs (t, u) where u is a tuple of ordinals below α.

Algorithm 4.4 (Seyfferth)**.**

```
set $\alpha = 0$;
MAIN:
set $t = ()$;
set $u = ()$;
set $n = 0$;
call $\mathrm{DFS}(t, u)$;
increment $\alpha$;
call MAIN;
$\mathrm{DFS}(t, u)$:
if $n = \omega$ then stop;
if $u(n) = \alpha$ then set $u(n) = 0$ and increment $t(u)$;
if $t(n) = \omega$ then decrement $n$ and set $t = t \upharpoonright n$ and set $u =$
$u \upharpoonright n$;
```

```
if $(x \restriction n, t, u) \in S$ then increment $n$ and set $t = t \frown 0$ and
set $u = u \frown 0$ and call $\mathrm{DFS}(t,u)$ and decrement $n$ and set
$t = t \restriction n$ and set $u = u \restriction n$;
increment $u(n)$ and call $\mathrm{DFS}(t,u)$;
```

Here in every call of $\mathrm{DFS}(t,u)$, the algorithm tries to extend t and u simultaneously by all pairs (m,β) with $m \in \omega$ and $\beta < \alpha$. Again, if $(t \frown m, u \frown \beta) \in S_x$, the sequence is immediately extended further and $\mathrm{DFS}(t \frown m, u \frown \beta)$ is called. Otherwise, $(t \frown m, u \frown \beta + 1)$ is tried next. If for all $\beta < \alpha$ $(t \frown m, u \frown \beta)$ cannot be extended further, then $(t \frown m + 1, u \frown 0)$ is tried next, and so on.

Lemma 4.5 (Seyfferth). *The algorithm will find the lexicographically least z such that $S_{x,z}$ has a branch, and the lexicographically least branch v through $S_{x,z}$, if such a real z exists.*

Proof. Assume z and v are as required. As in Lemma 4.3, we can see that before stage γ (where γ is the supremum of the range of the embedding coded by v), only countably many nodes are visited. In stage γ, only countably many nodes are visited before the branch (z,v) is found. □

It is straightforward to generalize this algorithm to semi-decide Σ^1_2 subsets of ${}^k({}^\omega\omega)$.

Remark 4.6. Notice that the halting time of any halting OTM computation with input a real x is countable: If we collapse a countable elementary substructure of some $L_\alpha[x]$ which contains the computation as an element, the collapsing function maps the computation to an initial segment, since OTM computations are absolute between transitive models of KP (see [Koe05, Lemma 2.6]). So the computation in fact halts at a countable time.

Proposition 4.7 (Seyfferth). *The OTM semi-decidable subsets of ${}^k({}^\omega\omega)$ are exactly the Σ^1_2 sets. The OTM decidable sets are the Δ^1_2 sets.*

Proof. The Shoenfield tree and Lemma 4.5 prove that all Σ^1_2 sets are OTM semi-decidable. On the other hand, OTM semi-decidable sets are easily seen to be Σ^1_2 definable. The second statement follows. □

If one wants to prove Shoenfield absoluteness without referring to the Shoenfield tree (which is defined in terms of descriptive set theory), it can be replaced with a tree defined by only computational means. We will describe such a tree in Remark 4.23. It also can replace the Shoenfield tree in all following proofs.

From the algorithms, we obtain short proofs of several results in classical descriptive set theory (cf. [Jec03, Kec95]).

Corollary 4.8 (Seyfferth). *Suppose M is a transitive model of KP with $\omega_1 \subseteq M$. Then Σ^1_2 relations are absolute between M and V.*

Proof. Since OTM computations are absolute between transitive models of KP, so is membership in Σ^1_2 sets. □

Corollary 4.9 (Schlicht-Seyfferth). *Every Σ^1_2 binary relation on the reals has a Σ^1_2 uniformization and every Π^1_1 binary relation on the reals has a Π^1_1 uniformization.*

Proof. Suppose $A \subseteq {}^{\omega}\omega \times {}^{\omega}\omega$ is a Σ^1_2 set. The algorithm semi-deciding $(x, y) \in A$ can be modified to search for a y given x as input. As we added the search for sequences $t \in {}^{<\omega}\omega$ to Algorithm 4.2 to obtain Algorithm 4.4, we may also add another search for $s \in {}^{<\omega}\omega$ with $(s, t, u) \in S_x$. An argument analogous to Lemmas 4.3 and 4.5 proves that the lexicographically least branch (y, z, v) through S_x is found. This corresponds to the lexicographically least branch through $S_{x,y}$, therefore $(x, y) \in A$. For any Π^1_1 binary relation, a similar modification of Algorithm 4.2 yields an algorithm semi-deciding a uniformization such that for any pair (x, y) in the uniformizing function, the algorithm halts before the least (x, y)-admissible ordinal $\omega_1^{x,y}$ above ω. Hence the uniformization is Π^1_1 by the Spector-Gandy theorem. □

This immediately implies:

Corollary 4.10 (Schlicht-Seyfferth)**.** *Every nonempty Σ^1_2 set of reals has a Σ^1_2 member, i.e. some x such that $\{x\}$ is a Σ^1_2 set, and every nonempty Π^1_1 set of reals has a Π^1_1 member.*

Remark 4.11. The proof of Corollary 4.9 shows that any function from the reals to the reals with OTM semi-decidable graph is OTM computable. This is false in general, e.g. when we consider OTM programs P such that $P(x)$ halts before ω_1^x for all x with $P(x) \downarrow$. Let us consider a Π^1_1 function f, obtained via Π^1_1 uniformization, mapping a real x to a code for a wellfounded countable model containing x of the theory T, where T is the extension of KP requiring that there is an admissible ordinal. Although its graph is semi-decidable by such a program, it is easy to see that f is not OTM computable by a program of this type.

Corollary 4.12 (Schlicht-Seyfferth)**.** *Every Σ^1_2 set is the union of ω_1 many Borel sets.*

Proof. Given a Σ^1_2 set A, let P be an OTM which terminates on input x if and only if $x \in A$. Let A_β denote the set of reals x such that $P(x)$ terminates before stage β. Then A is the union of the sets A_β. To see that each A_β is Borel, let a_β be a real coding the supremum γ_β over the halting times of the algorithm if restricted to at most β stages.[2] Then a real x is an element of A_β if and only if for some (for every) real c coding a computation along a_β, this computation halts. This shows that A_β is Δ^1_1 and hence Borel by Suslin's theorem. □

Corollary 4.13 (Schlicht-Seyfferth)**.** *Every Σ^1_2 set has a Σ^1_2 norm.*

Proof. Let A be a Σ^1_2 set and let P be an algorithm semi-deciding A. The desired norm is given by the map ϕ where P halts at time $\phi(x)$ on input x. Let $x \leq y$ ($x < y$) if $P(x)$ halts (strictly) before $P(y)$, or $P(x)$ halts and $P(y)$ does not halt. Then $y \in A$ and $x \leq y$ imply $x \in A$. Using the algorithm, it is easy to see that the relations $\leq$ and $<$ are OTM semi-decidable, hence Σ^1_2. We can define a Π^1_2 relation $\leq'$ by $x \leq' y \leftrightarrow \neg y < x$ which coincides with $\leq$ on A and again $y \in A$ and $x \leq' y$ implies $x \in A$. Hence ϕ is a Σ^1_2 norm on A. □

[2]If the algorithm terminates in stage β, the machine halts after at most $(\omega^\omega \cdot \beta^\omega) \cdot \beta$ many steps.

Note that we cannot obtain a Σ^1_2 norm whose initial segments are uniformly Borel. This would imply the existence of an uncountable sequence of distinct Borel sets of bounded rank, but it is known that this does not follow from ZF [Har78, Theorem 4.5].

In order to describe the supremum of the ordinals appearing as the halting time of some OTM program, let δ^1_2 denote the supremum of lengths of Δ^1_2 wellorders on sets of natural numbers. Let $\delta^1_2(x)$ denote the supremum of the length of Δ^1_2 wellorders in the parameter x on sets of natural numbers. Note that a real x is Δ^1_2 if and only if $\{x\}$ is Δ^1_2 or even just Σ^1_2.

Corollary 4.14 (Schlicht-Seyfferth)**.** *The supremum of halting times of OTMs with input x is $\delta^1_2(x)$.*

Proof. Suppose y codes a Δ^1_2 wellorder in the parameter x of type γ. Since y is OTM computable, we consider the algorithm which searches through the well-order given by y. The algorithm halts at a time at least γ.

Conversely, let us consider the Π^1_1 set in the parameter x of pairs (y, z) such that y codes a wellorder w with a maximal element l and domain the natural numbers and z codes a halting computation along w on input x which halts at l. This set contains a Π^1_1 singleton (y, z) in the parameter x by Corollary 4.9. Then y codes a Δ^1_2 wellorder in the parameter x whose order type is the length of the computation. □

4.3 Tree representations from computations

In this section, we construct a tree representation for an OTM semi-decidable set of reals from finite fragments of OTM computations. The tape content over an entire halting OTM computation on countable input by a program P can be viewed as an $\omega_1 \times \omega_1$ matrix filled with zeroes and ones. Every row represents the tape content at a given time. If we add a state and a head position per row, the computation is entirely captured in the resulting diagram:

tape $\rightarrow$

		state	head	0	1	2	3	4	5	6	7	8	9	$\cdots$	ω	$\cdots$
	0	0	0	0	0	0	0	0	0	0	0	0	0	$\cdots$	0	$\cdots$
	1	1	1	1	0	0	0	0	0	0	0	0	0	$\cdots$	0	$\cdots$
	2	0	2	1	0	0	0	0	0	0	0	0	0	$\cdots$	0	$\cdots$
	3	1	3	1	0	1	0	0	0	0	0	0	0	$\cdots$	0	$\cdots$
	4	0	4	1	0	1	0	0	0	0	0	0	0	$\cdots$	0	$\cdots$
	5	1	5	1	0	1	0	0	0	0	0	0	0	$\cdots$	0	$\cdots$
time $\downarrow$	6	0	6	1	0	1	0	1	0	0	0	0	0	$\cdots$	0	$\cdots$
	7	1	7	1	0	1	0	1	0	0	0	0	0	$\cdots$	0	$\cdots$
	8	0	8	1	0	1	0	1	0	1	0	0	0	$\cdots$	0	$\cdots$
	9	1	9	1	0	1	0	1	0	1	0	0	0	$\cdots$	0	$\cdots$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		
	ω	0	ω	1	0	1	0	1	0	1	0	1	0	$\cdots$	0	$\cdots$
	$\omega+1$	1	$\omega+1$	1	0	1	0	1	0	1	0	1	0	$\cdots$	1	$\cdots$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		

We will approximate similar diagrams by adding single bits of information. A *tape bit* $(\alpha, \beta, c, \lambda)$ will consist of:

1. a coordinate (α, β) in the $\omega_1 \times \omega_1$ matrix representing time α and tape cell β
2. the cell content $c \in \{0, 1\}$
3. a countable limit ordinal (or zero) λ – this number will be used to control the limit behavior.

Per row we also need a *machine bit* $[\alpha, s, \gamma, \lambda]$ containing the following information:

1. some time α, corresponding to the row in the matrix
2. a machine state s of P
3. a head position $\gamma \in \omega_1$
4. a countable limit ordinal (or zero) λ – this number will be used to control the limit behavior.

We will use the symbol '$\cdot$' if we do not want to specify a certain component of a tape or machine bit in the argument at hand (i.e. $(0, n, c, \cdot)$). A finite set of tape and machine bits can be coded into a countable ordinal; fix such a coding. We will now define the tree T, depending on the program P, on $\omega \times \omega_1$: A pair $(t, u) \in {}^k\omega \times {}^k\omega_1$ is in T if and only if

1. The set coded by u_j contains the bits coded by u_i for $0 \leq i \leq j < n$.
2. Every u_i contains at most one machine bit for each α and at most one tape bit for every pair of α and β.
3. For every tape bit $(0, n, c, \cdot)$ of u_i with $n < k$, we have that $t = s_k$, i.e. t serves as the initial segment of the initial tape contents of the partial computation.
4. u_0 contains a machine bit of the form $[0, 0, \cdot, \cdot]$ and a tape bit of the form $(0, 0, 0, \cdot)$. Also it contains a machine bit $[\alpha, s, \gamma, \cdot]$ plus a tape bit $(\alpha, \gamma, c, \cdot)$ where P does not contain an instruction for machine state s and currently read symbol c, i.e. α is a halting time. So the beginning and the end of the partial computation are fixed.
5. As soon as we have information about a tape cell at time α, we also know the machine state and head position: If u_i contains a tape bit $(\alpha, \cdot, \cdot, \cdot)$, it also contains a machine bit $[\alpha, \cdot, \cdot, \cdot]$.
6. We always know the tape cell to be read by the read-write head: If u_i contains a machine bit $[\alpha, \cdot, \gamma, \cdot]$, it contains a tape bit $(\alpha, \gamma, \cdot, \cdot)$.
7. If u_i contains a tape bit $(\alpha, \beta, c, \cdot)$, u_{i+1} contains bits immediately above (only if α is a successor ordinal) and below along the time axis: Let $[\alpha, s, \gamma, \cdot]$ be the corresponding machine bit given by rule 5. If $\beta = \gamma$ we require u_{i+1} to contain a tape bit $(\alpha + 1, \gamma, \cdot, \cdot)$ and a machine bit

$[\alpha+1,\cdot,\cdot,\cdot]$ as required by the program P. If α is a successor, we also similarly require the tape and machine bit of the form $[\alpha-1,\cdot,\cdot,\cdot]$ that P implies. Except for those tape bits, all the other tape cells should not change their content, so we add tape bits $(\alpha+1,\beta,c,\cdot)$ if $\beta \neq \gamma$. Again, if α is a successor, we add such tape bits $(\alpha-1,\beta,c,\cdot)$ for all β but the one for which we already added such a bit according to P.

8. For tape bits of limit times we have to ensure that the tape contents are inferior limits over earlier times: If λ is a limit ordinal, and $(\lambda,\beta,c,\cdot)$ is a tape bit of u_i. Suppose $c=0$. Then there is a tape bit $(\alpha,\beta,0,\cdot)$ with $\alpha<\lambda$ in u_{i+1} and $\alpha>\alpha'$ for all bits $(\alpha',\beta,\cdot,\cdot)$ in u_i with $\alpha'<\lambda$. If $c=1$ then there is a tape bit $(\alpha,\beta,1,\lambda)$ in u_{i+1} with $\alpha<\lambda$ and where α is larger than any time of a similar bit in u_i. Let α' be minimal such that u_{i+1} contains a tape bit of the form $(\alpha',\beta,1,\lambda)$. Then every tape bit for tape cell β and time $\bar{\alpha}$ between α and λ in u_{i+1} must be of the form $(\bar{\alpha},\beta,1,\cdot)$.

9. We also want the machine state at limit times to be a lim inf: If λ is a limit ordinal and u_i contains a machine bit $[\lambda,s,\cdot,\cdot]$, u_{i+1} contains a machine bit $[\alpha,s,\cdot,\lambda]$ where $\alpha<\lambda$ and where α is larger than any time of a similar bit in u_i. Let α' be minimal such that u_{i+1} contains a machine bit of the form $[\alpha',s,\cdot,\lambda]$. Then every machine bit for time $\bar{\alpha}$ between α and λ in u_{i+1} must be of the form $[\bar{\alpha},s',\cdot,\cdot]$ where $s'\geq s$.

10. Finally, we want to make the head position at limit times a lim inf as in the definition of OTMs. If λ is a limit ordinal, then for every machine bit $[\lambda,s,\gamma,\cdot]$ of u_i one of the following conditions hold: Either there is a machine bit $[\alpha,s,\gamma,\lambda]$ in u_{i+1} with $\alpha<\lambda$ where α is larger than any time of a similar bit in u_i and for every machine bit in u_{i+1} of the form $[\alpha',s,\gamma',\cdot]$ where α' is between α and γ we have $\gamma'\geq\gamma$. Or, alternatively, u_{i+1} does not contain a bit of the form $[\alpha,s,\gamma,\lambda]$, then we require that there is a bit $[\alpha,s,\gamma',\lambda]$ in u_{i+1} that is not in u_i where $\gamma'<\gamma$ and γ' is greater or equal to any $\gamma''<\gamma$ in any bit of u_{i+1}.

This means that every entry of the matrix given by u_i is extended both up- and downwards along the time axis in u_{i+1} while respecting the behavior of the program P and the limit rules involved in the definition of OTMs.

Let us, in the following, write $\operatorname{dom}(u_i)$ for the set of α such that u_i contains a bit of the form $(\alpha,\cdot,\cdot,\cdot)$. Moreover, let $\operatorname{dom}(u)=\bigcup_{i\in\omega}\operatorname{dom}(u_i)$.

Lemma 4.15 (Schlicht-Seyfferth). *T projects to the set of reals semi-decided by P.*

Proof. First let x be semi-decidable by P, i.e. $P(x)\downarrow$. We will show how to use the halting computation C to find a branch of T_x. Let $(\lambda_i)_{i\in\omega}$ be an enumeration of the limit times involved in C. Let $[\lambda_i,s_i,\gamma_i,\cdot]$ be the corresponding machine bits, and $(\lambda_i,\gamma_i,c_i,\cdot)$ the corresponding tape bits according to C, for $i\in\omega$. We can make sure that u_i contains both $[\lambda_i,s_i,\gamma_i,\cdot]$ and $(\lambda_i,\gamma_i,c_i,\cdot)$ and tape and machine bits $[\alpha,\cdot,\gamma,\cdot]$, $(\alpha,\gamma,\cdot,\cdot)$ with $\lambda_m<\alpha<\lambda_n$ for any $m<n<i$. Let us close $(u_i)_{i<\omega}$ under above rules using bits compatible with C. It is clear that for any two consecutive limits λ_k and λ_l, there is some u_i which contains bits

$(\alpha, \cdot, \gamma, \cdot)$, $[\alpha, \gamma, \cdot, \cdot]$ with $\lambda_k < \alpha < \lambda_l$. Since all bits are chosen form C, the gaps between the λ_i can be filled and $(u_i)_{i<\omega}$ forms a branch in T_x.

Now let $(u_i)_{i\in\omega}$ be a branch of T_x. We need to prove that the computation C by P on input x halts. Let $(\lambda_i)_{i\in\omega}$ be an enumeration of the limits in $\mathrm{dom}(u)$.

Claim 4.16. The ordinals in $\mathrm{dom}(u)$ are exactly the ordinals $\lambda_j + n$ for $j, n \in \omega$.

Proof of Claim. By the rules above it is clear that every ordinal of the form $\lambda_j + n$ is in $\mathrm{dom}(u)$. Suppose that μ is a limit and $\mu + n \in \mathrm{dom}(u_i)$ where $\mu \neq \lambda_j$ for all $j \in \omega$. Then it follows from the rules that $\mu \in \mathrm{dom}(u_{i+n})$, a contradiction. □

The set of bits in $(u_i)_{i\in\omega}$ induce a partial matrix U of the type pictured above. We call a submatrix *according to* P, if the machine state, head position, and tape contents change only as dictated by P.

Claim 4.17. For $\lambda \in (\lambda_i)_{i\in\omega}$ the submatrix of U induced by the rows $\lambda + n$ for all $n \in \omega$ is according to P.

Proof of Claim. Let $n \in \omega$ and choose i minimal such that $\exists m \lambda + m \in \mathrm{dom}(u_i)$. The rules dictate that, for any $m \in \omega$, $u_{i+|n-m|}$ contains unique machine bits for all rows between $\lambda+m$ and $\lambda+n$. Those machine bits and also the tape contents covered by bits present in u_i are changed only according to P. Of course, new tape cells might have been introduced by tape bits in u_j, $j > i$. But for any such given tape cell β, its content is kept constant except for actions of P. If at any stage a new bit would have been required to be added that conflicts with bits already present in u, the branch would not have been extended further. □

It remains to show that at limit times, machine state, head positions, and tape contents are inferior limits.

Claim 4.18. Let λ be in $(\lambda_i)_{i\in\omega}$. Let $(\alpha_j)_{j<\nu}$ be an increasing enumeration of $\mathrm{dom}(u) \cap \lambda$. Then:

(i) For every tape bit $(\lambda, \beta, c, \cdot)$, c is the inferior limit over the d in tape bits of the form $(\alpha_j, \beta, d, \cdot)$ in $\bigcup_{i\in\omega} u_i$.

(ii) For every machine bit $[\lambda, s, \cdot, \cdot]$, s is the inferior limit over the r in machine bits of the form $[\alpha_j, r, \cdot, \cdot]$ in $\bigcup_{i\in\omega} u_i$.

(iii) For every machine bit $[\lambda, s, \gamma, \cdot]$, γ is the inferior limit over the δ in machine bits of the form $[\alpha_j, s, \delta, \cdot]$ in $\bigcup_{i\in\omega} u_i$, if this $\liminf$ is a head position occurring in $\bigcup_{i\in\omega} u_i$, or γ is the least head position occurring in $\bigcup_{i\in\omega} u_i$ that is greater than the $\liminf$.

Proof of Claim. (i) Choose u_i such that $(\lambda, \beta, c, \cdot)$ is in u_i. Let $((\alpha_k, \beta, d_k, \cdot))_{k\in\mu}$ be an increasing (in α_k) enumeration of the tape bits in $(u_j)_{i<j<\omega}$ where $\alpha_j < \lambda$. First consider $c = 0$. The rules imply that $(d_k)_{k\in\mu}$ contains an unbounded sequence of 0s, hence c is in fact the inferior limit. Now suppose $c = 1$. In u_{i+1} a tape bit of the form $(\alpha, \beta, 1, \lambda)$ is added and all d_k where $\alpha_k > \alpha$ are ≥ 1.

(ii) Choose u_i such that $[\lambda, s, \cdot, \cdot]$ is in u_i. Let $([\alpha_k, s_k, \cdot, \cdot])_{k\in\mu}$ be an increasing (in α_k) enumeration of the machine bits in $(u_j)_{i<j<\omega}$ where $\alpha_j < \lambda$. In u_{i+1} a machine bit of the form $[\alpha, s, \cdot, \lambda]$ is added, where α is greater than any time of a similar bit in u_{i+1}. Indeed in every u_j where $j > i$ such a bit is added, so $(s_k)_{k<\mu}$ contains s unboundedly often. Also, the rules imply that every $s_k \geq s$ for for all $\alpha_k \geq \alpha$.

(iii) Choose u_i such that $[\lambda, s, \gamma, \cdot]$ is in u_i. Let $([\alpha_k, s, \gamma_k, \cdot])_{k\in\mu}$ be an increasing (in α_k) enumeration of the machine bits in $(u_j)_{i<j<\omega}$ where $\alpha_j < \lambda$ (note that we only consider bits with machine state s).

Case 1. In u_{i+1} a machine bit of the form $[\alpha, s, \gamma, \lambda]$ is added, where α is greater than any time of a similar bit in u_{i+1}. Indeed in every u_j where $j > i$ such a bit is added, so $(\gamma_k)_{k<\mu}$ contains γ unboundedly often. Also, the rules imply that every $\gamma_k \geq \gamma$ for for all $\alpha_k \geq \alpha$.

Case 2. No such bit is added in any u_j, $i < j$. Then by the rules, $(\gamma_k)_{k\in\mu}$ is strictly increasing below γ. Note that by the rules there is no head position in u that is between $sup_{k\in\mu}(\gamma_k)$ and γ. So even if $\liminf_{k<\mu}(\gamma_k) < \gamma$, the partial computation behaves as if γ was indeed the lim inf. □

□

We can alter rule 1 in the definition of the tree of partial computations to have the nodes u_i contain information about when which bits where added, allowing u_i to be decoded into $(u_j)_{j<i}$. Let us assume this extra requirement for the next lemma. We consider the lexicographical well-order $<_{lex}$ between bits. If $d = \{d_i : i \leq m\}$ and $e = \{e_i : i \leq n\}$ are finite sets of bits with $d_0 <_{lex} ... <_{lex} d_m$ and $e_0 <_{lex} ... <_{lex} e_n$, we define $d <_{lex} e$ if $|d| < |e|$, or $|d| = |e|$ and $d_i <_{lex} e_i$ for the least i with $d_i \neq e_i$. If u, v both satisfy the properties of the sequence u in the definition of T, we can decode sequences $u_0, u_1, ..., u_m = u$ and $v_0, v_1, ..., v_n = v$ from u and v such that for all $i < m$, u_i extends to u_{i+1} by a set of additional bits, which we call u^+_{i+1}, as stated by the rules for T, and similarly v_j for $j < n$. Let us define $u <_{tree} v$ by $m < n$, or $m = n$ and $u^+_i <_{lex} v^+_i$ for the least i with $u_i \neq v_i$.

Lemma 4.19 (Schlicht-Seyfferth)**.** *T has pointwise leftmost branches with respect to $<_{tree}$, i.e. that for every input x on which the computation halts, the tree T_x has a branch b so that $b_n \leq_{tree} c_n$ for every branch c of T_x and for every n..*

Proof. Let us consider the computation with input x. Let

$$b_0 = \{[0, 0, 0, 0], (0, 0, 0, 0), [\alpha, s, \gamma, 0], (\alpha, \gamma, c, 0)\},$$

where α is the halting time, s is the machine state at time α, γ is the head position at time α, and c is the content of cell γ at time α. Let b^+_{n+1} be $\leq_{lex}$-least such that the extension b_{n+1} of b_n by b^+_{n+1} describes a fragment of the given computation and is in T. Suppose towards a contradiction that c is a branch in T_x and n is minimal with $c_n <_{tree} b_n$. We can recover the predecessors $b_0, ..., b_{n-1}$ of b_n and $c_0, ..., c_{n-1}$ of c_n. If $b_i <_{tree} c_i$ for some $i < n$, then $b_n <_{tree} c_n$, contradicting the choice of n. Hence $b_i = c_i$ for all $i < n$ by minimality of n. This implies $b^+_n \leq_{lex} c^+_n$ by the definition of b and thus $b_n \leq_{tree} c_n$, contradicting the assumption. □

Remark 4.20 (Schlicht-Seyfferth). A stricter variation of T would only allow extensions by a minimal number of bits necessary to fulfill the conditions, and require the bits to be chosen $<_{lex}$-minimal in the sense that an extension of a node may not add $<_{lex}$-smaller bits which satisfy the same requirement as a given bit. If we consider this variation and change the definition of T so that u_i consists only of the additional bits relative to $\bigcup_{j<i} u_j$, it is not hard to see that T has pointwise leftmost branches with respect to $<_{lex}$.

Remark 4.21 (Schlicht-Seyfferth). The tree T induces a Σ^1_2 scale on the set of reals semi-decided by P. Let $x \leq_n y$ if both $P(x)$ and $P(y)$ halt and $b_x(n) \leq_{tree} b_y(n)$, for the leftmost branches $b_x \in T_x$ and $b_y \in T_y$, respectively, or $P(x)$ halts and $P(y)$ does not halt. The relation $x <_n y$ has an analogous definition with $\leq_{tree}$ replaced by $<_{tree}$. To prove that T is the tree from a scale, it is sufficient to show that the relations $\leq_n$ and $<_n$ induced by T are OTM semi-decidable. We semi-decide $x \leq_n y$ by simulating P on the inputs x and y, as in Corollary 4.13. If $P(x)$ halts before $P(y)$, we halt the program. If $P(x), P(y)$ halt at the same step, we run an OTM computation to determine whether $b_x(n) \leq_n b_y(n)$ and halt the program, if this is the case. Otherwise we let the program diverge. The argument for $<_n$ is similar.

Remark 4.22 (Schlicht). As a natural extension of the scale property, we might ask for a tree T projecting to a Σ^1_2 universal set A such that T_x has a unique infinite branch for every $x \in A$. Let us argue that the existence of such a tree is not provable in ZF. Assuming such a tree T exists, let $S = \{(s, ((s_0, t_0), ..., (s_{n-1}, t_{n-1}))) : n \in \omega, (s,t) \in T\}$. Then $A = p[T] = p[S]$ and there is a unique $b_x \in S_x$ for every $x \in A$, and $b_x \neq b_y$ for all $x \neq y$. Since for each $\alpha < \omega_1$ the projection of S restricted to ordinals below α is an injective image of a closed set and hence Borel, there is an n such that the set B of values of $b_x(n)$ for $x \in A$ is unbounded in ω_1. Let us choose the leftmost branch (x_α, b_α) in S with $b_\alpha(n) = \alpha$ for each $\alpha \in B$. We have defined an uncountable sequence $(x_\alpha : \alpha \in B)$ of distinct reals. However, there is no such sequence in the symmetric model for the Levy collapse $Col(\omega, < \aleph_\omega)$, as was pointed out to the authors by Daisuke Ikegami.

It is possible to prove Shoenfield absoluteness without referring to tree representations from descriptive set theory, using only computational means:

Remark 4.23 (Schlicht-Seyfferth). The tree of partial computations may be used instead of the Shoenfield tree to show that every Σ^1_2 set is OTM semi-decidable. We can see that $\Sigma^1_1(x)$ sets are ordinal semi-decidable via a depth-first search for a witness for the $\Sigma^1_1(x)$ statement. The length of this search is bounded by the least α such that $L_\alpha[x]$ is a model of KP and Σ_1-separation (see [Bar75, Theorem 9.6]), and this ordinal α is computable on input x by recursively writing codes for $L_\beta[x]$ for increasing β while checking the axioms. Hence Σ^1_1 and Π^1_1 sets are OTM decidable. To semi-decide a Σ^1_2 set $A = p[B]$ with B in Π^1_1, we now search on input x for a real y and a branch in the tree of partial computations for B on input (x, y).

4.4 Applications

In [Koe05, Definition 1] the *programs* that steer the computations of OTMs are defined with the following condition: If the machine is currently in state s

and the machine's read-write head currently reads symbol c, then the program contains at most one command for that situation. This way, when Koepke defines the *ordinal computation* by a program P, he can refer to the unique command in a given situation. Instead, for the present section, we shall drop the above restriction on programs and define ordinal computations in a way that, in successor steps, the lexicographically least instruction (if there is one that suits the current situation) is chosen to determine the next machine step. This allows us to define *non-deterministic* ordinal computations as follows.

Definition 4.24 (Seyfferth)**.** Given program P and an input (i.e. an initial tape configuration), the *non-deterministic ordinal Turing computation (NOTM computation)* by P is defined like the ordinal computation by P ([Koe05, Definition 2]), except that in successor steps any suitable command may define the machine's next step.

NOTM computations may be used to define sets of reals.

Definition 4.25 (Seyfferth)**.** A set of reals $A \subseteq {}^{\omega}\omega$ is *NOTM semi-decidable* if there is a program P such that

$$x \in A \leftrightarrow \text{ there is a halting NOTM computation by } P \text{ on input } x$$

Consider a countable substructure of a transitive set containing such a computation as an element. Then the image of the computation under the collapsing map is a countable halting NOTM computation by P on input x.

As in the case of classical Turing decidability, given a coding of the "choices" that a NOTM computation makes, NOTM decidability can be verified deterministically:

Lemma 4.26 (Seyfferth)**.** *There is a program Q such that for every program P and every real input x, there is a real z such that the OTM computation by Q on inputs P, x, and z halts if and only if some NOTM computation by P on input x halts.*

Proof. Let us define z to code two reals z_1 and z_2. Let z_1 code a well-order on ω of order type the (countable) length of the NOTM computation by P on input x. Let z_2 be such that in machine step $\mathrm{otp}_{z_1}(i)$, the OTM computation by P on input x selects the $z_2(i)$-th least command P contains for that situation. Note that both otp_{z_1} and $\mathrm{otp}_{z_1}^{-1}$ are OTM computable functions. Now the program Q is essentially a universal OTM which selects the $z_2(i)$-th command in P in the $\mathrm{otp}_{z_1}(i)$-th simulation step. □

This settles the question of whether NOTMs compute more sets of reals than OTMs:

Proposition 4.27 (Seyfferth)**.** *Every NOTM (semi-)decidable set of reals is already OTM (semi-)decidable.*

Proof. Let $A \subseteq {}^{\omega}\omega$ and suppose that Q is the program from Lemma 4.26. Then A is NOTM semi-decidable if and only if there is a program P so that for every input x there is a real z such that the OTM computation by Q on inputs P, x, and z halts. Since this is a Σ^1_2 statement, A is Σ^1_2 and hence OTM semi-decidable. □

Note now that the existence of certificates z established in Lemma 4.26 is absolute. Thus, if there are any certificates, there is one in L. Using Shoenfield absoluteness we could search for such a z through L, using the OTM computable recursive truth predicate from [Koe05]. Instead, let us search for a certificate via the tree of partial computations:

Lemma 4.28 (Seyfferth)**.** *Given a program P and an element (s, u) of the full tree on $\omega \times \omega_1$, we can OTM decide the question of whether or not (s, u) is an element of the tree of partial computations according to P.*

Proof. We first have the OTM check whether u codes a set of tape and machine bits. If yes, we can easily check the finitely many conditions (rules 1-10) if u is a partial computation by P on some input that is compatible with s. □

With the preceding lemma, we can use a variant of Algorithm 4.4 to find branches in the tree of partial computations. Since Propositions 1 and 2 hold also for our algorithm operating on the tree of partial computations, we get:

Proposition 4.29 (Seyfferth)**.** *There is an algorithm such that, if A is NOTM semi-decidable by the program P, then, given x as an input, the algorithm will find a real $z \in {}^{\omega}\omega$ such that the OTM computation by Q (cf. Lemma 4.26) on inputs P, x, and z halts if $x \in A$ and diverges otherwise.*

Proof. If x is in A, there is a z in L such that the OTM computation by Q on inputs P, x, and z halts. An argument analogous to Propositions 4.3 and 4.5 shows that given a real x, a straighforward adaptation of Algorithm 4.4 will find a branch of the form (x, c) in the tree T of partial computations by P, if any exists. From c the desired z can be easily decoded. □

The tree representation allows us to generalize the results in Section 4.2 to sets of reals semi-decided by ordinal machines with upper bounds on the halting times.

Definition 4.30 (Schlicht-Seyfferth)**.** Suppose f is a function from the reals to the ordinals. Let us say that a set of reals A is f-semi-decidable or Γ_f if there is a OTM program P semi-deciding A such that P halts before time $f(x)$ on input x if it halts at all.

For our purpose, we are interested in functions of the following form:

Definition 4.31 (Schlicht-Seyfferth)**.** Suppose f is a function from the reals to the ordinals. We call f *multiplicatively closed* if $f(x) \leq f(\langle x, y\rangle)$ for all reals x and y and $f(x)$ as an ordinal is closed under ordinal multiplication. Let us call f *admissible* if furthermore $f(x)$ is x-admissible for all reals x.

The classes Γ_f for admissible f with values strictly above ω range from Π^1_1 to Σ^1_2. Recall the definition of δ^1_2 from the paragraph before Corollary 4.14.

Lemma 4.32 (Schlicht-Seyfferth)**.** *Let $f(x) = \omega_1^x$ and $g(x) = \delta^1_2(x)$. Then Γ_f is the class of Π^1_1 sets and Γ_g is the class of Σ^1_2 sets.*

Proof. Suppose that A is f-semi-decidable via P and x is a real. Then $x \in A$ if and only if in every countable model of KP, every computation by P with input x halts. Since such models can be coded into reals, A is Π^1_1. Suppose A is Π^1_1

and $x \in A$ if and only if T_x is wellfounded. Then $rank(T_x) < \omega_1^x$ and hence the algorithm searching for a branch in the Shoenfield tree halts before ω_1^x. The statement for Σ_2^1 sets follows from Corollary 4.14. □

Corollary 4.33 (Schlicht-Seyfferth)**.** *Suppose f is multiplicatively closed. Then every Γ_f set has a Γ_f norm.*

Proof. Suppose a set in Γ_f is semi-decidable by a program P with halting time bounded by f. Let $\phi(x)$ be the halting time of P on input x. Since f is multiplicatively closed, ϕ is a Γ_f-norm as in the proof of Corollary 4.13. □

Corollary 4.34 (Schlicht-Seyfferth)**.** *Suppose f is admissible. Then every Γ_f binary relation has a uniformization with graph in Γ_f.*

Proof. Suppose a relation in Γ_f is semi-decidable by a program P with halting time bounded by f. We apply the algorithm for searching through the Shoenfield tree to the tree of partial computations. Let us consider a program R for the variant of Algorithm 4.4 which on input (x, y) searches for a real z and a branch in the tree of halting computations of P with input (x, z). We claim that if the program finds such a pair, then this happens before the time $\alpha = f(\langle x, z\rangle)$. Let us assume towards a contradiction that z and the corresponding computation of P are found at a time $\gamma \geq \alpha$. If $\gamma = \alpha$, we map each $n \in \omega$ to the time at which $z \upharpoonright n$ appears first in the search, and thus obtain a $\Sigma_1^{L_\alpha[x,y]}$ definable cofinal map $h : \omega \to \alpha$. If $\gamma > \alpha$, there is a $t \in {}^{<\omega}\omega$ which appears first at the time α, and we obtain a $\Sigma_1^{L_\alpha[x,y]}$ definable cofinal map $h : \{s \in {}^{<\omega}\omega : s <_{lex} t\} \to \alpha$ by mapping s to the time at which it first appears in the search. This contradicts the $\langle x, z\rangle$-admissibility of α.

At this point, we let R halt if $y = z$ and let R diverge otherwise. For any real x in the domain of the relation, let $(g(x), b(x))$ be $\leq_{lex}$-least such that $b(x)$ is a branch in the tree of partial computations with input $(x, g(x))$. Then $R(x, g(x))$ halts and $R(x, z)$ diverges for all $z \neq g(x)$. Hence the graph of g is in Γ_f. □

Let us now consider ordinal machines with a set of reals as oracle as in [HL00]. In a query state in a computation, the program asks whether the sequence on the initial segment of length ω of the tape is an element of the set. Let us write $P^A(x)$ for the OTM computation by the program P with oracle A on input x. Let us also fix a computable enumeration $(P_n \mid n \in \omega)$ of all programs.

Definition 4.35 (Schlicht-Seyfferth)**.** The *halting problem relative to a set of reals A* or *jump of A* is defined as $A^\blacktriangledown = \{(n, x) \mid P_n^A(x)\downarrow\}$.

The *halting problem* $0^\blacktriangledown$ is a Σ_2^1 set, in fact we have:

Proposition 4.36 (Schlicht-Seyfferth)**.** *The halting problem $0^\blacktriangledown$ is Σ_2^1 universal. If $n \geq 1$ and $V = L$, then the n^{th} iterated jump $0^{\blacktriangledown n}$ is Σ_{n+1}^1 universal.*

Proof. Every halting computation with countable input halts at a countable time (see Remark 4.6). Hence $(m, x) \in 0^{\blacktriangledown n}$ is described by a Σ_{n+1}^1 formula stating the existence of a wellorder w on the natural numbers with largest element l together with a sequence indexed by w, coding a computation of P_m with input x and oracle $0^{\blacktriangledown(n-1)}$ halting at l. Let us suppose that A is defined by the formula $\exists x \varphi(x, y)$, where φ is Π_n^1. We consider a program searching through L for a witness for φ as in [Koe05], using the oracle $0^{\blacktriangledown(n-1)}$ to verify $\varphi(x)$ for reals x. This program identifies A as a section of $0^{\blacktriangledown n}$. □

In particular, the Σ^1_{n+1} sets are exactly the OTM semi-decidable sets in a Σ^1_n oracle for $n \geq 1$, if $V = L$. Let us show that this remains true when $\kappa \geq \omega_1$ many Cohen reals are added to L by the forcing $Add(\omega, \kappa)$.

Lemma 4.37 (Schlicht)**.** *Suppose that $V = L[G]$, where G is $Add(\omega, \kappa)$-generic over L, and $\kappa \geq \omega_1$. Then $0^{\blacktriangledown n}$ is Σ^1_{n+1} universal for all $n \geq 1$.*

Proof. Suppose that x is a real in $L[G]$. There are an $Add(\omega, 1)$-generic filter g_0 with $L[x] = L[g_0]$ and an $Add(\omega, \kappa)$-generic filter g_1 over $L[g_0]$ with $L[G] = L[g_0][g_1]$. Let us consider the case $n = 2$ and suppose φ is a binary Π^1_2-formula. Then $\exists y \varphi(x, y)$ holds in $L[G]$ if and only if $\exists \sigma \in N \Vdash_{Add(\omega,1)} \varphi(y, \sigma)$ holds in $L[x]$, where N is the set of nice $Add(\omega, 1)$-names for reals in $L[x]$. Since nice names for reals are coded by reals, this is a Σ^1_3 statement in $L[x]$. We can now express any Σ^1_{n+1} statement about x in $L[G]$ by a Σ^1_{n+1} statement in $L[x]$ uniformly in x for all $n \geq 1$ in a similar fashion. This is proved by induction on n. Every such set is a section of $0^{\blacktriangledown n}$ by the proof of the previous proposition. □

It is also consistent with ZFC that the iterated jumps have a lower complexity. Note that the assumption that $\omega_1^{L[x]} < \omega_1$ for every real x may be obtained by forcing with the Levy collapse $Col(\omega, < \kappa)$ below an inaccessible cardinal κ.

Lemma 4.38 (Schlicht)**.** *Suppose that $\omega_1^{L[x]} < \omega_1$ for every real x. Then $0^{\blacktriangledown n}$ is a Δ^1_3 set for all $n \geq 1$.*

Proof. Let us consider the Π^1_2 set A of pairs (x, y) such that y codes $L_\gamma[x]$ and γ is the least x-admissible ordinal above $\omega_1^{L[x]}$. We can compute the truth value of $x \in 0^{\blacktriangledown n}$ in $L_\gamma[x]$ using an algorithm which has access to n distinct tapes of length $\omega_1^{L[x]} + 1$. The original program runs on tape n. Whenever the oracle $0^{\blacktriangledown i}$ is called on tape i for $1 < i \leq n$, the oracle is computed on tape $i - 1$ by a subroutine of length $\omega_1^{L[x]} + 1$. The Π^1_2 description of A yields a Δ^1_3 description of the set of pairs (x, n) with $x \in 0^{\blacktriangledown n}$. □

The two previous lemmas imply that the complexity of $0^{\blacktriangledown n}$ is independent of the size of the continuum for $n \geq 2$.

4.5 Further questions

A set of reals A is said to be Σ^1_2 in a countable ordinal α if there is a Σ^1_2 formula $\varphi(x, y)$ such that for all reals y coding α and all reals x, $x \in A$ if and only if $\varphi(x, y)$ holds, i.e. the Σ^1_2 definition is independent of the coding of α. We leave open whether the sets of reals with a Σ^1_2 definition in an ordinal α, evaluated in $V^{Col(\omega,\alpha)}$, are exactly the OTM semi-decidable sets of reals with parameter α, and whether sets in these classes can be uniformized by functions with graphs in these classes.

Part III

Infinite time Blum-Shub-Smale machines

Chapter 5

Introduction to Blum-Shub-Smale machines

5.1 Blum-Shub-Smale machines

Computation on reals is a lesser known-branch of computability theory. Different approaches rival, one taking approximation as in numerical analysis as the defining concept [Wei00], the other taking points on the real line $\mathbb{R}$ as elementary. The latter concept, introduced in [BSS89] by Lenore Blum, Mike Shub, and Stephen Smale, lacks the capability for definition via limits. We set out to equip these machines with such a feature in Chapter 6.

We give a short account of the definition of a finite dimensional BSS machine on $\mathbb{R}$ as in [BSS89]. A function $\mathbb{R}^m \to \mathbb{R}^n$ is called *polynomial / rational* if every projection $f_i : \mathbb{R}^m \to \mathbb{R}$ is given by some polynomial in m variables / given by the fraction of two polynomials in m variables. The finite dimensional Blum-Shub-Smale programs for computations on $\mathbb{R}$ are usually presented as a flow chart, i.e., a connected digraph with node set $N = \{1, 2, \ldots, n\}$ and edge set $E \subseteq N \times N$. Let us fix $\mathbb{R}^p$ as the *input space*, $\mathbb{R}^m$ as the *state space*, and $\mathbb{R}^n$ as the *output space*. Such a digraph is a BSS program if there is an assignment $f : N \to \{\phi \mid \phi : R^k \xrightarrow{\text{rational}} R^l \wedge k, l \in \{p, m, n, 1\}\} \times \{0, 1, 2, 3\}$ such that

- There is exactly one $i \in N$ such that $f(i) = (\phi, 0)$.
- If $f(i) = (\phi, 0)$ then ϕ is linear and maps the input space to the state space and i has only one outgoing edge and no incoming edges. We call i the *input node*.
- If $f(i) = (\phi, 1)$ then ϕ is linear and maps the state space to the output space and i has no outgoing edges. We call i an *output node*.
- If $f(i) = (\phi, 2)$ then ϕ maps the state space to the state space and i has precisely one outgoing edge. We call i a *computation node*.
- If $f(i) = (\phi, 3)$ then ϕ maps the state space into $\mathbb{R}$ and i has two outgoing edges. We call i a *branch node*.

Computations are carried out by following the data through the flowchart, starting from the unique input node. At input, computation, and output nodes, the data is transformed according to the respective functions and transferred along the unique edges to the next nodes. At branch nodes, the data is unchanged. However, depending on the node's function's value on the data, the data is passed on to the edge leading to the node with lower index if the value is less than zero, and to the one with greater index otherwise. Without changing the result of valid computations, we insert new branch nodes before every computation and branch nodes to intercept denominators becoming zero and directing the data to a specific new output node that will be interpreted as 'no valid computation exists'.

In the original paper, an infinite dimensional case is also studied, for which the distinction between input, output, and state space becomes important. For the purpose of this thesis, the finite dimensional case is sufficient, hence we be omitting input and output space in the sequel without any loss of generality. In the next chapter, we will give a presentation of the definition more akin to register machines with n registers in cases where the state space is $\mathbb{R}^n$.

In the following, we shall give a brief summary of Chapter 6 published as [KS12].

5.2 Summary of the next chapter

The idea of a strict-limit-based BSS model of ordinal computation has been floating around in the ordinal computability community for some time. Koepke and the author decided it was both worth fleshing out and a suitable aspect for the present thesis.

The paper is built around a definition made rigorous by the author. The definition itself is based on a modified BSS definition that does away with the nonstandard flow-chart intuition of the original definition [BSS89] and introduces a more register machine-like notation. The author is quite certain that such a presentation has been given before, but it seemed sensible to use a notation that ties in neatly with previous work on ORMs and ITRMs by Koepke et al. A noteworthy aspect of this is to leave out arbitrary real coefficients and restrict the atomic functions used in the BSS computations to rational functions with only rational coefficients: In the classical BSS context, this would appear as an arbitrary limitation and somewhat goes against the idea of using the reals $\mathbb{R}$ in their algebraic sense. However, once generalized to transfinite computations, the difference between real and rational parameters becomes drastic: The binary expansion of a real can code set theoretically very powerful objects. This would lead to the computations being heavily dependent on the underlying model of set theory. The general direction of ordinal computability, however, is to base considerations on very absolute notions of computability and only then, and in a hopefully controlled manner, adjoin parameters that may introduce behavior of higher set theoretical impact.

The modified BSS definition is then expanded to give the notion of *infinite time Blum-Shub-Smale machines (ITBMs)* by adding as a limit rule the requirement that, at limit time, all register contents need to converge and are set to their respective limits. Note that strict limits in $\mathbb{R}$ (as opposed to $\liminf$'s) are used for the real register contents. The set of program instructions is finite, so, in this case, the $\liminf$ is the appropriate choice for the program instruction called at the limit time.

The paper goes on to give some examples of elementary functions that are not BSS but ITBM computable (exponential, sine, and cosine). It is noted that, in all examples, some work is necessary to make registers containing scratch work converge in the limit. The question of a general procedure to make auxiliary registers converge uniformly is raised but not answered. Such a technique has been developed in the mean time in, as of yet, unpublished work by Peter Koepke and Andrey Morozov. The examples then given, showing that all ordinals below ω^ω can appear as halting times of ITBMs, prepare for the main result of the paper that every ITBM computation halts before ω^ω many steps or diverges. This conjecture is due to the author. The theorem was then proved using a lemma by Koepke after in-depth discussion with the author and can thus be attributed to them jointly. The following discussion of the strength of the ITBM model, incorporating comparisons with classical Turing machines and ITTMs is due to the author. Koepke provided a constructibility theoretic argument to show that ITBM computations can be carried out in the first ω^ω-many levels of the constructible hierarchy. The first open question given at the end of the paper has been answered in the aforementioned unpublished work by Koepke and Morozov: Every real in L_{ω^ω} is in fact ITBM computable. The proof works by showing that the iterated Turing jumps $\emptyset^\alpha$ for $\alpha < \omega^\omega$ are ITBM computable.

All reals in L_{ω^ω} are computable in some $\emptyset^\alpha$ and consequently ITBM computable. As was pointed out to the author by Philip Welch (cf. [Wel]), L_{ω^ω} is in also the least set that contains ω and is closed under the *safe set recursion* studied in [BBF12].

Chapter 6

Towards a theory of infinite time Blum-Shub-Smale machines [KS12] [1]

[1] in contrast to the published version, the version printed here includes authorship tags for theorems, definitions, etc.

6.1 Introduction

In the spirit of ordinal computability — the study of classical models of computations generalized to transfinite ordinal numbers — we study a variation of the Blum-Shub-Smale (BSS) machine introduced in [BSS89]. In contrast to established models of ordinal computability, such as Hamkins' and Lewis' infinite time Turing machines (ITTMs) [HL00] and Koepke's ordinal Turing machines (OTMs) [Koe05], these machines employ real numbers in the classical continuum $\mathbb{R}$ as opposed to elements of Baire space ${}^{\omega}\omega$ or Cantor space ${}^{\omega}2$. The topological differences matter as soon as we consider limits (see below). Variations thereof, be it in allowing infinitely many registers or changes in the limit behavior, might very well change the computational strength. In this paper, we aim for the "weakest" possible generalization of BSS machines into ordinal time. We believe that already this restricted model shows interesting properties.

Our machines have a finite number n of *registers*, each containing a real number. Generalizations to other fields and rings are possible but shall not be of concern to this paper. The computation is steered by a finite program $P \subseteq \omega \times \{f \mid f : \mathbb{R}^n \xrightarrow{\text{rational}} \mathbb{R}^n\} \times \{0,1\} \times \omega \times \omega$, containing commands of the form (i, ϕ, j, k, l), where i is the index of the command at hand, ϕ is a rational map (with rational coefficients), and j tells us if the command represents a *computation node* or a *branch node*. In case $j = 0$, we are at a computation node, the register content $x \in \mathbb{R}^n$ is replaced by $\phi(x)$ and the next command (index $i + 1$) is carried out next. The values of k and l are ignored in this case. Otherwise, $j = 1$ and we are at a branching node. This means that the register content is left unchanged and, depending on whether $\phi(x) > 0$, the next command will be the one with index k. If on the other hand $\phi(x) \leq 0$, command number l is carried out next. We can assume the indices of a given program's commands to form an initial segment of the natural numbers and that no index appears twice. In case a command index is called for which no command in the program exists, the computation halts.

Note 6.1. As a minor technical detail we would like to note that, as in the original paper [BSS89], we avoid discontinuity points of rational functions by putting decision nodes before each computation or decision node to check whether the denominator of the rational function to be evaluated is 0. If not, we continue as planned, if yes, an infinite loop is entered and the computation diverges.

So far, we have outlined a standard BSS machine with the additional restriction that the rational functions present in computation and branching nodes do not allow for arbitrary real coefficients. We add irrational coefficients in form of *parameters* later on. We now make our machines access the transfinite: In order for the machine to run for infinitely many steps, we have to *define* the register content at limit times. In the established theories of infinite time or ordinal Turing and register machines, often an inferior or superior limit is used for this purpose. Instead, we want to restrict ourselves here to ordinary limits of sequences of real numbers. This immediately implies that there will be situations where an infinite time BSS machine will, e.g., be properly defined at any finite time but not at the first limit time ω because the register contents do not converge. We can imagine the machine to "crash" in such a case and say that for such a combination of program and input no valid computations exists. In case of converging register contents we also have to come up with a command that is carried out at a limit time. For this we will indeed use the inferior limit, i.e., the command with the least index that was used cofinally often below the limit. Note that we do not introduce a dedicated limit state.

Let us put things together in the following definition.

Definition 6.2 (Seyfferth)**.** Let $n \in \omega$ be a number of registers. Let $k < n$ and let P be a $n+k$-register BSS program. The *infinite time BSS machine computation (ITBM computation) with parameters* $p_1, p_2, \ldots, p_k \in \mathbb{R}$ *by* P on some input $x \in \mathbb{R}^n$ is defined as the transfinite sequence $(C_t)_{t\in\theta} = (R(t), I(t))_{t\in\theta} \in {}^{\theta}(\mathbb{R}^{n+k} \times \omega)$ where:

(a) $\theta \in \mathrm{Ord}$ or $\theta = \mathrm{Ord}$

(b) $R(0) = (x, p_1, p_2, \ldots, p_k)$ and $I(0) = 0$

(c) *(computation node)* If $t < \theta$ and $I(t) = i$ let $(i, \phi, 0, k, l)$ be the command of P with index i. Then $R(t+1) = \phi(R(t))$ and $I(t+1) = i+1$.

(d) *(branching node)* If $t < \theta$ and $I(t) = i$ let $(i, \phi, 1, k, l)$ be the command of P with index i. Then $R(t+1) = R(t)$ and if furthermore $\phi(R(t)) > 0$ then $I(t+1) = k$; if on the other hand $\phi(R(t)) \leq 0$, then $I(t+1) = l$.

(e) If $t < \theta$ is a limit and $y = lim_{s\to t} R(s)$, then $R(t) = y$ and $I(t) = \liminf_{s\to t} I(s)$.

(f) If $\theta <$ Ord, then θ is a successor ordinal and $I(\theta - 1)$ calls a command index that is not in P (the machine *halts (in θ-many steps)*).

We define ITBM computable functions on the reals:

Definition 6.3 (Seyfferth)**.** A function $f : \mathrm{dom}\, f \to Y$ where $\mathrm{dom}\, f, Y \subseteq \mathbb{R}$ is called *ITBM computable in parameters* $p_1, p_2, \ldots, p_k$ if there is an at least $k+1$-register ITBM program P s.t. for every $x \in \mathrm{dom}\, f$ the computation by P on input $(x, 0, 0, \ldots, 0, p_1, p_2, \ldots, p_k)$ halts and the final register content is of the form $(f(x), \cdot, \cdot, \ldots, \cdot)$. On input $x \notin \mathrm{dom}\, f$ the computation is required to diverge. We call such a function *ITBM computable* if $k = 0$, i.e., no real parameters are necessary.

The use of one limit step enables us to compute the classical elementary functions that are defined by power series as illustrated by the following examples. While such functions as the exponential function can be computed in classical recursive analysis, they are not computable in the standard BSS model [Bra03].

Example 6.4 (Seyfferth)**.** The exponential function $e : \mathbb{R} \to \mathbb{R}, x \mapsto e^x = \sum_{k=0}^{\omega} \frac{x^k}{k!}$ is ITBM computable: Define a 5-register program that computes the desired function if $|x| < 1$. Later we shall describe the modifications necessary to work for any x.

Algorithm 6.5 (Seyfferth)**.**

```
input $R_1 = x$;
set $R_2 := R_3 := R_4 := R_5 := 1$; (initialize)
call loop;
```

```
loop:
if $R_2 = 0$ then set $R_1 := R_5$ and halt, else continue;
set $R_4 := \frac{R_4}{R_4+1}$; (store $\frac{1}{i}$, where $i$ is the current iteration)
set $R_3 := R_3 * R_4$; (store $\frac{1}{i!}$)
set $R_2 := R_2 * R_1$; (store $x^i$)
set $R_5 := R_5 + \frac{\frac{1}{R_2}}{R_3}$; (store $\frac{i!}{x^i}$)
call loop;
```

If $|x| < 1$, all register contents converge and R_5 contains the desired output at time ω, which is correctly recognized when $R_2 = 0$. We can adapt the algorithm for $|x| \geq 1$ by adding a case distinction in the beginning and, in case $|x| \geq 1$, save $\frac{1}{x^i}$ in register three. Then register three converges also at limit times. Of course, the command that updates R_5 inside the loop has to be changed accordingly.

Example 6.6 (Seyfferth)**.** The sine function *sin* : $\mathbb{R} \to \mathbb{R}, x \mapsto \sum_{k=0}^{\omega}(-1)^k \frac{x^{(2k+1)}}{(2k+1)!}$ and the cosine function *cos* : $\mathbb{R} \to \mathbb{R}, x \mapsto \sum_{k=0}^{\omega}(-1)^k \frac{x^{2k}}{(2k)!}$ are ITBM computable. This is proven by the previous example and the fact that $(-1)^k$ can be recovered from $(-\frac{1}{2})^k$ and $(\frac{1}{2})^k$, both of the latter which can be convergently stored and updated in separate registers.

Note 6.7. Common to this type of examples is that some tricks are necessary to make all registers converge at limits. Auxiliary registers used for scratch work often do not contain converging content. If their content is bounded, however, one can simply divide the register content cofinally often by a fixed number and keep track of how often this division has occurred. Unbounded contents are best stored as their multiplicative inverse. Both approaches can be imagined as pushing the relevant data contained in a register into increasingly later places in their decimal/binary expansion. Compound limits like $\omega \cdot \omega$ are an additional problem, as scratch registers cannot be set to arbitrary values after limit times without sacrificing convergence at the compound limit. However, in every limit stage towards a compound limit the register content will be bounded if treated like above. So, in order to ensure convergence at the compound limit, these bounds themselves need to converge. See the next chapter for an example.

6.2 Clockable ordinals

Since Hamkins' and Lewis' paper on ITTMs [HL00], determining those ordinals that appear as halting times on empty inputs has proved to be important for the study of machine models. Since our machines do not halt at limit times, we are interested in machines that run for some limit number α many steps and halt in the next step:

Definition 6.8 (Seyfferth)**.** An ordinal α is called *ITBM clockable* if there is an $n \in \omega$ and an n-register ITBM program that halts on input $0 \in \mathbb{R}^n$ in exactly $\alpha + 1$ many steps.

The algorithms above prove that ω is clockable, but let us establish this anew with an algorithm that uses only one register.

Lemma 6.9 (Seyfferth)**.** *The first limit ordinal ω is ITBM clockable.*

Proof. By algorithm.

Algorithm 6.10 (Seyfferth)**.**

```
set $R_1 := 1$;
call loop;
loop :
if $R_1 = 0$ then halt else continue;
set $R_1 := \frac{R_1}{2}$;
call loop;
```

□

We can clock $\omega \cdot n$ by having n loops in separate lines of code, where loop1 calls loop2 and loop i calls loop $i+1$ instead of the halting command, each time resetting R_1 to 1. Instead, we could also use a separate register to perform a countdown from n. When trying to extend this approach trivially to clock $\omega \cdot \omega$, we run into a problem that is connected to the fact that $\omega \cdot \omega$ is a compound limit, i.e., a limit of limits: At time $\omega \cdot \omega$, R_1 will have cofinally often been set from 0 to 1, so convergence or R_1 fails. While this is easily fixed as seen below, it hints at the limitations imposed by the strict limit rule and why the supremum of ITBM clockable ordinals might be quite low in the ordinals.

Lemma 6.11 (Seyfferth)**.** *The first compound limit ordinal $\omega \cdot \omega$ is ITBM clockable.*

Proof. By algorithm.

Algorithm 6.12 (Seyfferth)**.**

```
set $R_1 := 1$;
set $R_2 := 1$;
call loop;
loop:
if $R_2 = 0$ then halt else continue;
if $R_1 = 0$ then set $R_2 := \frac{R_2}{2}$ and set $R_1 := R_2$ and continue,
else continue;
set $R_1 := \frac{R_1}{2}$;
call loop;
```

In this algorithm, R_1 is halved repeatedly to detect limits. Once a limit time is reached ($R_1 = 0$), R_2 is halved and R_1 is reset not to 1 but to the current value of R_2. Once R_2 hits 0, we have found the compound limit $\omega \cdot \omega$. At every limit, every register content converges. □

As before, it is easy to clock finite multiples of the form $\omega \cdot \omega \cdot n$. Also, if we extend the algorithm to use extra registers $R_3, R_4, \ldots, R_n$ in the same manner as we extended the ω-algorithm with R_2, we can in fact clock any finite power ω^n. However, this is as far as we can get, as ω^ω turns out to be the supremum of the ITBM clockable ordinals.

First observe that at any limit time, the register contents are a fixed point for every command that has been carried out cofinally often:

Lemma 6.13 (Koepke)**.** *Let $(C_t)_{t<\theta}$ be an ITBM computation by some program P and let $\alpha < \theta$ be a limit ordinal. Then the first command in the computation that alters the register contents after time α has not been carried out cofinally often below α.*

Proof. Let $(c, \phi, 0, \cdot, \cdot) \in P$ be a computation node which is called cofinally often below α. Let c be called at some time $\beta > \alpha$, where for all $\alpha < \gamma < \beta$ the register content has not been changed yet, i.e., $R(\alpha) = R(\gamma) = R(\beta)$. Since ϕ may be assumed as locally continuous (cf. Note 6.1), we get:

$$\begin{aligned}
R(\beta+1) &= \phi(R(\beta)) \\
&= \phi(R(\alpha)) \\
&= \phi(\lim_{t\to\alpha} R(t)) \\
&= \lim_{t\to\alpha \wedge I(t)=c} \phi(R(t) \\
&= \lim_{t\to\alpha \wedge I(t)=c} R(t+1) \\
&= R(\alpha)
\end{aligned}$$

So the first computation node that changes the register content after time α cannot have been called cofinally below α. □

Theorem 6.14 (Koepke-Seyfferth)**.** *Let P be a program with k computation nodes. Then in any computation $(C_t)_{t<\theta}$ according to P, the register contents stabilize before ω^{k+1}.*

Proof. If $k = 0$ then the computation halts after finitely many steps or diverges since the program contains only finitely many branch nodes: The computation may run through these nodes in a finite sequence or in an infinite loop. Every branch node may trigger halting depending on its rational function evaluated on the unchanged input. After finitely many steps, every node in the sequence or loop has been visited once. If the computation didn't halt up to this point, the program will go on forever as the register content is never changed.

So let the hypothesis hold for k. Let P be a program with $(k+1)$-many computation nodes. Suppose the register contents change after ω^{k+2}. The first computation node c responsible for a new register content is not used cofinally often below ω^{k+2}. Let α be the supremum over the times $< \omega^{k+2}$ when c was carried out nontrivially. We can view $(C_t)_{\alpha \le t < \omega^k+2}$ as the computation by $P \setminus \{c\}$ on input $R(\alpha)$. Inductively, the register content of this computation stabilizes in ω^{k+1}-many steps. Since $\alpha + \omega^{k+1} < \omega^{k+2}$ this means that the original computation stabilizes before ω^{k+2}. But a computation that stabilizes before a limit can never change its register content again. □

Once the register contents have stabilized, an ITBM computation diverges or may run for an additional finitely many steps before halting. So we get:

Theorem 6.15 (Koepke-Seyfferth)**.** *The supremum of ITBM clockable ordinals is ω^ω.*

The above argument is in fact independent of the input:

Corollary 6.16 (Koepke-Seyfferth). *Every ITBM computation halts before ω^ω-many steps or diverges.*

Corollary 6.17 (Koepke-Seyfferth). *If an ITBM computation diverges, the register content at time ω^ω is not changed any more and can be considered the* pseudo-output *of the diverging computation.*

6.3 Connections to other models of computation

From a computability perspective, reals provide ample possibilities as codes for complex objects. We can code and decode into reals with our ITBMs by interpreting the binary expansion of a real in the interval $[0,1]$ as an element of the Cantor space ${}^\omega 2$, i.e., an ω-long sequence of 0's and 1's. The binary expansion of a real is not necessarily unique, so two binary strings representing the same real will appear to our machines as equivalent.

Let us give an algorithm to retrieve the n-th binary digit b_n of an $x = 0.b_0b_1b_2\cdots \in [0,1)$.

Algorithm 6.18 (Seyfferth).

```
input $R_1 = x$;
input $R_2 = n$;
set $R_3 = \frac{1}{2}$; ( $= 2^0$)
set $R_4 = 0$; (the current approximation to $x$)
call loop;
loop:
if $R_2 := 0$ then call lastloop else continue; if $R_4 + R_3 >$
$R_1$ then continue (do not add $R_3$ to the approximation $R_4$ if
the result would exceed $x$)
else set $R_4 := R_4 + R_3$ and continue; (add $R_3$ to the approximation)
set $R_3 := \frac{R_3}{2}$;
set $R_2 := R_2 - 1$; ($=$ #remaining_iterations)
call loop;
lastloop:
if $R_4 + R_3 > x$ then set $R_1 := 0$ and halt else set $R_1 := 1$
and halt;
```

With this algorithm we can also do local changes to the binary bits of x in the fashion of a Turing machine. So finite Turing computations can obviously be implemented on BSS/ITBMs. Also, the halting problem for Turing machines becomes ITBM computable:

Lemma 6.19 (Seyfferth). *The classical halting problem is ITBM computable.*

Proof. Since standard Turing computations are ITBM computable, we can generate Turing programs successively. So, in iteration n carry out the first n Turing programs for n many steps on empty input, using a dedicated simulation register. In step n, use only the n-th and later binary digits for the Turing simulation, so at time ω, this register will have converged to 0 (cf. Note 6.7).

Once the algorithm finds that some program (say, the i-th) halts on its input, change the i-th binary digit of an initially zero output register to 1. Since there is a finite time when all halting computations of programs with index $< n$ will have halted, this register converges to the halting problem. □

So our machines have more computing strength than Turing machines or classical BSS machines. Also, the type-2 Turing machines of computable analysis (see [Wei00]) can easily be simulated by ITBMs: The output tape of a type-2 Turing machine, when modeled as an ITBM register, converges by definition. Input tapes do not change their content and the finitely many work tapes can be made convergent like in Note 6.7.

Using the above coding, ITBMs can also operate on functions. A continuous function $f : \mathbb{R} \to \mathbb{R}$ may be input to an ITBM as its restriction to the rationals $f^{\mathbb{Q}} = f \upharpoonright \mathbb{Q} : \mathbb{Q} \to \mathbb{R}$ coded into a real $p_f \in [0, 1]$. This requires a fixed enumeration of rational numbers $q : \omega \to \mathbb{Q}$ which may be chosen as ITBM computable and an ITBM computable bijective pairing function $\langle \cdot, \cdot \rangle : \omega \times \omega \to \omega$. Then $f^{\mathbb{Q}}(x) = y$ may be expressed as "the $\langle q^{-1}(x), i\rangle$-th binary digit of p_f is exactly the i-th digit of y". By computing a convergent sequence of rationals for a given real (nested intervals) we can compute the function value at this real. This takes ω-many steps: Produce, for all $n < \omega$, the approximations of x up to n binary digits in a similar way to Algorithm 6.18 as a sequence of rationals converging to x. For every such approximation, decode from p_f the function value up to n digits. This defines a sequence of rationals that converge to $f(x)$.

Example 6.20 (Seyfferth)**.** The derivative of a differentiable function is ITBM computable.

Proof. Given a point x, have an ITBM evaluate the differential quotient in x using only $f(x)$ itself and rational approximations to $f(x)$. □

An upper bound on strength of ITBMs is given by the strength of ITTMs:

Lemma 6.21 (Seyfferth)**.** *Let P be an n-register BSS program. There is an ITTM program Q and a map $f : \mathbb{R}^n \to {}^{\omega}2$ s.t. for every $x \in \mathbb{R}$ the ITTM computation by Q on input $f(x)$ halts and returns the information that either no computation by P on x exists, or that P halts on x with output $y \in {}^{\omega}2$, or that the P diverges on x with pseudo-output $y \in {}^{\omega}2$, where $f^{-1}(y)$ is the final register content of the ITBM computation by P on x.*

Proof. We assume that the ITTM we are working with has a finite number of read-write tapes, which can be accomplished of interlacing these tapes onto the single scratch tape in the definition of ITTMs in [HL00]. Due to Corollary 6.17, we know that after time ω^ω also a diverging ITBM computation does not change its register content anymore. Since it is easy for an ITTM to construct a well order of ω of length ω^ω on one tape, it is possible for an ITTM to code the complete ITBM computation $(R_t, I_t)_{t<\omega^\omega}$ up to time ω^ω on the output tape.

So let us begin with defining a map $f' : \mathbb{R} \to {}^{\omega}2$. First, imagine the value $f'(x)$ to be contained in three elements of ${}^{\omega}2$ (i.e., three Turing tapes) where the first tape contains only a 0 or 1 in the first cell to specify the sign of x, the second contains the maximal exponent n s.t. $2^n \leq x$, and the third contains the binary expansion of $\frac{x}{2^n}$, ignoring the decimal (binary) point and normalized in the following way: Binary expansions that end on an infinite trail of 1s are replaced by the unique binary expansion of the same number that ends on a trail of 0s. Then, use an ITTM computable pairing function to interlace these three tapes into one. The function f' can easily be extended to a function $f : \mathbb{R}^n \to {}^{\omega}2$.

It is easy to see that an ITTM is perfectly capable — albeit with an enormous time consumption — of computing addition, subtraction, multiplication and division on elements of ${}^{\omega}2$ and of normalizing the result in the way described above. So, given the input, the program Q will start carrying out the sequence of ITBM commands in P and writing the results (and the program instructions used) one after another in the respective cells of the output tape. At limit times, the output tape contains all the information of the previous times, so it is easy for the Q to check whether everything converges and compute the limit if one exists. If not, output that there is no valid computation by P on x. If yes, Q will continue its simulation of P up to time ω^ω. At time ω^ω, it can replace the output tape content with $f(y)$ where y is P's output or pseudo-output. □

It turns out that ITTMs are much stronger than needed, as will be clear from the following. Gödel's constructible model L of set theory is closely related to infinite time computations. We shall use Ronald Jensen's J_α-hierarchy to study definability in L. The $J_\alpha[\overrightarrow{x}]$-hierarchy relativized to the real parameters $\overrightarrow{x}$ is defined by the following recursion on the ordinals: $J_0[\overrightarrow{x}] = \emptyset$; $J_{\alpha+1}[\overrightarrow{x}]$ is the closure of $J_\alpha[\overrightarrow{x}] \cup \{J_\alpha[\overrightarrow{x}]\}$ under all rudimentary functions, using also the parameters $\overrightarrow{x}$. The rudimentary functions are simple set theoretic functions which include the formation of ordered pairs. Also first-order definitions over structures can be computed rudimentarily. Note that the ordinal height of $J_\alpha[\overrightarrow{x}]$ is $J_\alpha[\overrightarrow{x}] \cap \mathrm{Ord} = \omega \cdot \alpha$. In the sequel, the level $J_{\omega^\omega}[\overrightarrow{x}]$ will play a special role. It is also a member of the standard L-hierarchy, and indeed $J_{\omega^\omega}[\overrightarrow{x}] = L_{\omega^\omega}[\overrightarrow{x}]$ is the least level beyond ω where the two hierarchies coincide.

In the following we assume that real numbers are coded by their binary expansions. Then a real number a will be a function from ω into the set $\{0, 1, ., -\}$, where . denotes the binary dot and $-$ is the minus sign. The real a can be considered a subset of $H_\omega = J_1[\overrightarrow{x}]$. We show that an ITBM computation can be uniformly defined along the $J_\alpha[\overrightarrow{x}]$-hierarchy.

Lemma 6.22 (Koepke). *Let $(C_t)_{t\in\theta}$ be an ITBM computation according to a program P on input $\overrightarrow{x} \in \mathbb{R}^n$. Then for all $\alpha > 0$ the following hold:*

(i) If $t < \omega \cdot \alpha$ then $C \restriction t + 1 \in J_{1+\alpha}[\overrightarrow{x}]$.

(ii) $C \restriction \omega \cdot \alpha$ is uniformly $\Sigma_1(J_{1+\alpha}[\overrightarrow{x}])$ definable in the parameters $\overrightarrow{x}$ and H_ω.

Proof. By the set theoretic recursion theorem, Definition 1 yields a definition of $(C_t)_{t\in\theta}$ of the form

$$\begin{aligned} y = C_t \quad \leftrightarrow \quad & \exists f : t+1 \to V[f(0) = G_0(\overrightarrow{x}) \\ & \wedge \forall u < t(f(u+1) = G_1(f(u), H_\omega)) \\ & \wedge \forall u < t(limit(u) \to f(u) = G_2(f \restriction u, H_\omega)) \\ & \wedge y = f(t)]. \end{aligned}$$

The functions G_0, G_1, G_2 are rudimentary: G_0 produces the initial configuration C_0 from the input $\overrightarrow{x}$. This amounts to assembling $(R(0), I(0))$ from the components of $\overrightarrow{x}$. This operation is certainly rudimentary.

The function G_1 is the transition function from the configuration at time u to the configuration at time $u+1$. The transition involves some case distinctions and the application of rational functions to register contents. The complexity of real arithmetic is indeed arithmetical in the arguments: for reals a and b the relations $a = b$, $a < b$ and the reals $a + b$, $a \cdot b$, $a - b$, and $\frac{a}{b}$ (in case $b \neq 0$) are first-order definable over the structure (H_ω, a, b). Since $H_\omega = J_1[\overrightarrow{x}] \in J_2[\overrightarrow{x}]$, real arithmetic is rudimentary, using the extra parameter H_ω.

The function G_2 performs the ITBM limit operation at limit times $u < \theta$. Given $f \upharpoonright u$, the ordinary limit in the binary reals can be obtained by first-order quantification over the multisorted structure $(H_\omega, u, f \upharpoonright u)$. So G_2 is rudimentary.

This means that definition (1) is a Σ_1-definition of $y = C_t$ whose kernel is rudimentary. Rudimentary predicates are absolute with respect to any level of the $J_\alpha[.]$-hierarchy. Note that the unique witness for the existential quantifier in (1) is $C \upharpoonright (t+1)$.

We now show the lemma by simultaneous induction on $\alpha > 0$. By our considerations so far, (i) for α implies (ii) for α.

Case $\alpha = 1$. If $t < \omega \cdot \alpha = \omega$ then $C \upharpoonright t+1$ is built from $\overrightarrow{x}$ and H_ω by finitely many applications of the rudimentary functions G_0 and G_1. Since $J_{1+1}[\overrightarrow{x}]$ is closed under rudimentary functions, $C \upharpoonright t+1 \in J_{1+1}[\overrightarrow{x}]$.

Case $\alpha = \beta+1$, where the lemma holds for β. Then $C \upharpoonright \omega \cdot \beta$ is $\Sigma_1(J_{1+\beta}[\overrightarrow{x}])$ definable in parameters, and hence $C \upharpoonright \beta \in J_{1+\alpha}[\overrightarrow{x}]$. For $t \in [\omega \cdot \beta, \omega \cdot \alpha)$, $C \upharpoonright t+1$ can be built from $C \upharpoonright \beta$ by finitely many applications of the rudimentary functions G_0 and G_1. Hence $C \upharpoonright t+1 \in J_{1+\alpha}[\overrightarrow{x}]$.

For α being a limit ordinal, property (i) for all $\beta < \alpha$ immediately implies property (i) at α. □

Corollary 6.23 (Koepke). *Every ITBM computable real is an element of $L_{\omega^\omega} = J_{\omega^\omega}$.*

A set $A \subseteq \mathbb{R}$ is ITBM *decidable* if there is an ITBM program that outputs 0 on inputs $x \in A$ and 1 otherwise. A set $B \subset \mathbb{R}$ is ITBM *semi-decidable* if there is a program P that halts only on the $x \in B$.

Corollary 6.24 (Koepke). *All ITBM (semi-)decidable sets of reals lie in $L_{\omega^\omega}[\mathbb{R}]$.*

6.4 Remarks and open questions

As with other models of ordinal computability, we have established a connection between computability and Gödel's constructibility. A natural conjecture is that Corollary 3 can be reversed. Can the constructible hierarchy up to L_{ω^n} be "simulated" by an ITBM? This will require ITBMs to be able to do simple syntactic operations and inductions up to ω^n, for every natural number n.

ITBMs form "pointwise" limits at every limit time. This should lead to connections with the Baire hierarchy of functions.

Relaxing the limit rules by going to lim inf's, e.g., will allow computations to go on beyond time ω^ω and will lead to stronger notions of computability. In [Koe06] resp. [KM08], [CFK$^+$10] we studied machines with finitely many registers that contain natural numbers. At limit times register contents are the lim inf's of previous register contents. The weaker machines in [Koe06] crash when one of these $\liminf$'s is ∞ whereas in [KM08], [CFK$^+$10] that register is reset to 0. One can use similar limit rules for infinite time generalizations of Blum-Shub-Smale machines. Those generalizations obviously are able to simulate the machines of [Koe06] resp. [KM08], [CFK$^+$10] and should thus allow to compute all hyperarithmetic reals resp. finitely iterated hyperjumps.

Bibliography

[Bar75] Jon Barwise. *Admissible sets and structures.* Springer-Verlag, 1975.

[Bar81] Hendrik Pieter Barendregt. *The lambda calculus. Its syntax and semantics.* Studies in Logic and the foundations of mathematics. North/Holland publishing company, Amsterdam, New York, Oxford, 1981.

[BBF12] Arnold Beckmann, Samuel R. Buss, and Sy-David Friedman. Safe recursive set functions. 2012. Submitted for publication.

[Bra03] Vasco Brattka. The emperors new recursiveness: The epigraph of the exponential funciton in two models of computability. In Masami Ito and Teruo Imaoka, editors, *Words, languages and combinatorics III*, pages 63–72. World Scientific Publishing, Singapore, 2003.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, 1989.

[CFK$^+$10] Merlin Carl, Tim Fischbach, Peter Koepke, Russell Miller, Miriam Nasfi, and Gregor Weckbecker. The basic theory of infinite time register machines. *Archive for Mathematical Logic*, 49(2):249–273, 2010.

[Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[CR36] Alonzo Church and J. Barkley Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, 1936.

[Daw09] Barnaby Dawson. *Ordinal time Turing Computation.* PhD thesis, University of Bristol, 2009.

[Dev73] Keith J. Devlin. *Aspects of constructibility.* Lecture Notes in Mathematics. Springer-Verlag, Berlin, Heidelberg, 1973.

[Fis10] Tim Fischbach. The Church-Turing Thesis for Ordinal Computable Functions. Master's thesis, University of Bonn, October 2010.

[Har78] Leo Harrington. Analytic determinacy and $0^\sharp$. *J. Symbolic Logic*, 43(4):685–693, 1978.

[Hjo10] Greg Hjorth. Vienna notes on effective descriptive set theory and admissible sets. available at http://www.math.uni-bonn.de/people/logic/events/young-set-theory-2010/Hjorth.pdf, 2010.

[HL00] Joel D. Hamkins and Andy Lewis. Infinite time Turing machines. *The Journal of Symbolic Logic*, 65(2):567–604, 2000.

[Jec03] Thomas Jech. *Set theory.* Springer Monographs in Mathematics. Springer-Verlag, Berlin, 2003. The third millennium edition, revised and expanded.

[JK71] Ronald B. Jensen and Carol Karp. Primitive recursive set functions. In Dana S. Scott, editor, *Axiomatic Set Theory*, volume XIII (Part 1) of *Proceedings of Symposia in Pure Mathematics*, pages 143–176, Providence, Rhode Island, 1971. American Mathematical Society.

[Kan03] Akihiro Kanamori. *The higher infinite: large cardinals in set theory from their beginnings.* Springer monographs in mathematics. Springer-Verlag, 1994,2003. Second edition.

[Kec95] Alexander S. Kechris. *Classical descriptive set theory*, volume 156 of *Graduate Texts in Mathematics.* Springer-Verlag, New York, 1995.

[KM08] Peter Koepke and Russell Miller. An enhanced theory of infinite time register machines. In A. Beckmann et al, editor, *Logic and Theory of Algorithms*, volume 5028 of *Lecture Notes in Computer Science*, pages 306–315. Springer-Verlag, Berlin, Heidelberg, 2008.

[Koe05] Peter Koepke. Turing computations on ordinals. *The Bulletin of Symbolic Logic*, 11:377–397, 2005.

[Koe06] Peter Koepke. Infinite time register machines. In A. Beckmann et al, editor, *Logical Approaches to Computational Barriers*, volume 3988 of *Lecture Notes in Computer Science*, pages 257–266. Springer-Verlag, Berlin, Heidelberg, 2006.

[Koe09] Peter Koepke. Ordinal computability. In Klaus Ambos-Spies, Benedikt Löwe, and Wolfgang Merkle, editors, *Mathematical Theory and Computational Practice*, volume 5635 of *Lecture Notes in Computer Science*, pages 280–289. Springer-Verlag, Berlin, Heidelberg, 2009.

[KS06] Peter Koepke and Ryan Siders. Register computations on ordinals. *submitted to: Archive for Mathematical Logic*, 14 pages, 2006.

[KS09] Peter Koepke and Benjamin Seyfferth. Ordinal machines and admissible recursion theory. *Annals of Pure and Applied Logic*, 160(3):310–318, 2009. Computation and Logic in the Real World: CiE 2007.

[KS12] Peter Koepke and Benjamin Seyfferth. Towards a theory of infinite time Blum-Shub-Smale machines. In B. Cooper et al, editor, *How the world computes*, volume 7318 of *Lecture Notes in Computer Science*, pages 310–318. Springer-Verlag, Berlin, Heidelberg, 2012.

[Lev63] Azriel Levy. Transfinite computability. *Notices of the American Mathematical Society*, 10:286, 1963. Abstract.

[SS12] Philipp Schlicht and Benjamin Seyfferth. Tree representations and ordinal machines. *Computability*, 1(1):45–57, 2012.

[Tur36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936. A correction in the following volume, 544-546.

[Wec10] Gregor Weckbecker. Ordinal register machines and constructibility. Master's thesis, University of Bonn, July 2010.

[Wei00] Klaus Weihrauch. *Computable analysis. An Introduction.* Springer-Verlag, Berlin, Heidelberg, 2000.

[Wel] Philip Welch. Transfinite Machine Models. In *Turing Legacy Volume.* The Association for Symbolic Logic. To appear.