Priority arguments in transfinite computability theory

MSc Thesis *(Afstudeerscriptie)*

written by

Steef Hegeman
(born April 2nd, 1998 in Leiden, Netherlands)

under the supervision of **Prof.dr Benedikt Löwe**, and submitted to the Examinations Board in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam*.

Date of the public defense:	**Members of the Thesis Committee:**
September 26, 2023	Dr Benno van den Berg
	Prof.dr Benedikt Löwe
	Dr Lorenzo Galeotti
	Dr Franziska Jahnke

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Abstract

The priority method has been used to examine the structure of the semi-decidable degrees in computability theory. We discuss three classical results (the Friedberg-Muchnik theorem, the splitting theorem, and the thickness lemma) and discuss whether their proofs could be adapted to transfinite computability theory, specifically to the machine models of ITTMs, α-machines, and p-α-machines (α-machines equipped an extra ordinal parameter). We prove a splitting theorem for certain ITTM-semi-decidable sets—sufficient to conclude that there are infinitely many ITTM-semi-decidable degrees—and a thickness lemma for certain ITTM-semi-decidable sets of low degree. We sketch results for α-machines and p-OTMs (ordinal Turing machines with an extra parameter), among which a splitting theorem and a thickness lemma for p-OTMs.

Acknowledgements

This project has been a happy anchor through difficult times. I am grateful to Benedikt for his understanding, patience, and guidance, allowing me to work at my own pace while helping me structure my approach. I also want to thank Tanja for the periodic meetings, Benno for helping me navigate the MoL, as well as the other staff and students I have had the pleasure of interacting with. I thank Amber, my parents, my sister, and my friends.

Errata
(found between submission and defense)

In the proof of proposition 1.32, it is claimed that the set of non-accidentally-writable reals is ITTM-decidable. What is actually known, however, is that the set of (codes for) non-accidentally-writable *ordinals* is ITTM-decidable. This suffices for proving the results. (In particular, a set containing one writable real and the codes for non-accidentally-writable ordinals is iRE and iSD, but not iGB.)

In chapter 3, it is claimed for various models that, as a corollary of a splitting theorem and Dekker theorem, an infinite complete binary tree can be found in the semi-decidable degrees. However, what is obtained is not necessarily an embedding of such a tree, but an embedding of an infinite directed acyclic graph, which after removing some arrows is an infinite complete binary tree.* The main result of there being infinitely many semi-decidable degrees is unaffected.

Typographical errors of note appear in the statement of lemma 4.12—whose conclusion should be that $h \leqslant_{\mathrm{ITTM}} A$ instead of the other way around—and in corollary 4.20, where it is C' that should be decidable—not C.

*Roughly, the directed acyclic graph is an infinite union of finite stages constructed as follows. Nodes—semi-decidable degrees—are numbered as they are added. Every stage, the least node d that has not yet been split receives attention. It is first split on all other nodes (of which there are finitely many): for the least node $k \neq d$, split d on k, obtaining degrees $d_k, d'_k \leqslant d$ with $k \nleqslant d_k, d'_k$. (Note: d_k, d'_k are not necessarily incomparable). Then split d_k on the second node $k' \neq d$, obtaining a $d_{k'} \leqslant d_k \leqslant d$ with $k' \nleqslant d_{k'}$. After doing this for all nodes $k \neq d$, a degree $d' \leqslant d$ is obtained with $k \nleqslant d'$ for all nodes $k \neq d$. Then d' is split on itself to obtain incomparable $a, b < d' \leqslant d$ with $k \nleqslant a, b$ for all nodes $k \neq d$. These are the new nodes.

It is not excluded for the $a, b < d$ added when d receives attention that there is a $k \neq d$ so that $d \nless k$ and $a < k$ hold. Considering only the arrows between nodes d and the nodes a, b added when d receives attention, one obtains a complete infinite binary tree.

Contents

Introduction

The priority method is a proof technique for constructing semi-decidable sets that satisfy infinitely many interdependent requirements. This thesis concerns the generalization of this method to models of transfinite computability theory.

The arguments considered are, in increasing complexity, the Friedberg-Muchnik theorem, the splitting theorem, and the thickness lemma. The first is a finite injury argument showing that there are incomparable semi-decidable degrees, the second a finite injury argument that depends on a given semi-decidable set, showing that the non-trivial semi-decidable degrees can be split into two incomparable lower semi-decidable degrees, and the third is an infinite injury argument from which well-known infinite injury results such as the density theorem follow [23, section 3].

In [5, section 4], Hamkins and Lewis proved a Friedberg-Muchnik theorem for ITTMs. In [7] a Friedberg-Muchnik theorem is proved for ORMs, and it is shown that ORMs and OTMs simulate each other, effectively proving a Friedberg-Muchnik theorem for OTMs. P-α-machines are a machine model for α-recursion theory [12], for which amongst others a Friedberg-Muchnik theorem [16], splitting theorem [21], and density theorem [22] has been proved. However, the main notion of reduction used in α-recursion theory is stricter than the Turing reduction considered here [12, theorem 21]. Koepke and Seyfferth proved a Friedberg-Muchnik theorem for p-α-machines with respect to Turing reduction [12, section 4].

In this thesis, we discuss some of these results, and adapt them to prove a splitting theorem for ITTM-semi-decidable subsets of ITTM-decidable sets, sufficient for showing that there are infinitely many ITTM-semi-decidable degrees. We also prove a thickness lemma for the same sets of low degree. Furthermore, we sketch proofs of a Friedberg-Muchnik theorem for α-machines (without parameters), OTMs, and p-OTMs, a splitting theorem for p-OTMs, and a thickness lemma for p-OTMs. The thickness lemmas require a reinterpretation of "thick subset". Table 1 contains an overview of the arguments discussed.

The first chapter of the thesis defines well-known models of (transfinite) computability and lays the groundwork for doing priority arguments. Chapters 2, 3, and 4 are dedicated to one priority argument each, starting with the classical proof and then discussing generalizations.

Model	Generalizations of classical priority argument results		
	Friedberg-Muchnik	Splitting theorem	Thickness lemma*
ITTMs	Holds [5]	† (thm. 3.13, cor. 3.34)	† (thm. 4.15, cor. 4.20)
α-machines	Holds (thm. 2.22)	Unknown (sec. 3.4)	Unknown
OTMs	Holds [7] (sec. 2.3)	Unknown (sec. 3.4)	Unknown
p-OTMs	Holds (thm. 2.23)	Holds (thm. 3.40)	Holds (thm. 4.22)

Table 1: Generalizations of priority arguments discussed in the thesis. Here α refers to an admissible ordinal.

†A splitting theorem and thickness lemma were proved for certain ITTM-semi-decidable sets. We do not know whether this can be expanded.

*A direct translation of the thickness lemma, where $A \subseteq B$ is a thick subset if $B^{[e]} \setminus A^{[e]}$ is finite for all e below the space bound, does not hold for any transfinite machine model discussed (corollary 4.10). Results were proved by weakening this constraint appropriately (to α-finite differences).

Chapter 1

Transfinite computability theory

Contemporary transfinite computability theory is based on generalizations of machine models from ordinary computability. The models discussed herein are based on Turing machines [25].[1] Intuitively, these are devices attached to a tape containing symbols. They can read one symbol at a time and, based on what they read and their “state of mind” [25, p. 250], they write a different symbol, move to one adjacent to it, and alter their state of mind. Transfinite generalizations prescribe what happens after infinitely many such steps.

The content of this chapter is based on machine definitions in [6], [10], [12] and the reference work [1]. Reference material for computability theory, set theory, and α-recursion theory can be found in [13], [9], and [18, chapter VII] respectively. The presentation is inspired by [4, sections 3, 4] and [11, section 1].

1.1 Machine models

Let $\Sigma = \{0, 1\}$ be the set of *symbols* and $Q = \omega$ the set of possible *states* containing distinct $q_0, q_*, q_?$. Let $n \geq 1$ be a natural number and $\alpha \leq \beta \leq \mathsf{On}$ each either nonzero limit ordinals or the class On of ordinals.

Definition 1.1. A *tape* is a sequence $t : \alpha \to \Sigma$ of *cells* containing symbols.

Definition 1.2. An n-tape machine *program* is a finite partial function

$$m : Q \times \Sigma^n \to Q \times \Sigma^n \times \{\Leftarrow, \Rightarrow\}.$$

Its elements are called *instructions*.

Definition 1.3. An n-α-configuration is a triple in $(\Sigma^\alpha)^n \times \alpha \times Q$.

Definition 1.4. Given an *n-α-configuration* $c = (t, i, q)$, the result $s_m(c)$ of a *step* by an n-tape program m is defined if and only if $m(q, t(i)) = (q', s', d)$ is defined.

[1]There are also transfinite models based on other machine types, such as register machines. They are generally not equivalent with similar Turing machine based models.

It is then given by (t', i', q'), where t' is the result of replacing the ith column of t with s' (so that $t(n)(i) = s'(n)$), and i' is $i + 1$ if $d = \Rightarrow$ holds, $i - 1$ if $d = \Leftarrow$ holds and i is a successor, or 0 otherwise.

Definition 1.5. Given an n-tape $t \in (\Sigma^\alpha)^n$ and an n-tape program m, the *computation* of m on t is the longest sequence $\{c_\delta = (t_\delta, i_\delta, q_\delta) : \delta < B \leq \beta\}$ satisfying (for all $\delta < B$, $k < n$, and $\xi < \alpha$)

- $c_0 = (t, 0, q_0)$;
- $c_{\delta+1} = s_m(c_\delta)$;
- and for limits $\lambda > 0$
  - $t_\lambda(k)(\xi) = \liminf_{\delta<\lambda} t_\delta(k)(\xi)$; (a cell is 1 iff it converges to 1)
  - i_λ is $\liminf_{\delta<\lambda} i_\delta$ if this is below β, or 0; (the tape wraps around)
  - $q_\delta = q_*$ (the machine is in the limit state.)

If $B < \beta$, the computation is said to *halt*. Otherwise, it *diverges*.

Example 1.6 (Hamkins and Lewis [6, §3]). The computation of the machine

$$\begin{aligned}
(q_0, 0) &\mapsto (q_1, 1, \Rightarrow)\\
(q_0, 1) &\mapsto (q_1, 1, \Rightarrow)\\
(q_1, 0) &\mapsto (q_0, 0, \Leftarrow)\\
(q_*, 1) &\mapsto (q_1, 0, \Rightarrow)
\end{aligned}$$

on any tape proceeds as follows: it writes a 1 on the first cell, after which it starts moving back and forth between the first two cells. On limit steps, it finds itself on the first cell. If the first cell contains a 0, the machine halts. If not, it writes a 0 on the first cell, then changes it back to a 1 (this is called "flashing" the cell), and starts moving back and forth between the first two cells again. Considering the lim inf-rule, we see that the machine halts at time ω^2—provided $\omega^2 < \beta$. ⌟

Let $D \subseteq \Sigma^\alpha$ and $P \subseteq D$.

Definition 1.7. An *machine* is a pair (m, p) of a 3-tape program m and a parameter $p \in P$. In this context, the three tapes are called the parameter, input, and output tape respectively.

Machines (m, p) can be seen as partial functions $f : D \to D$ as follows: $f(x)$ is defined if and only if the computation of m on $(p, x, 000\ldots)$ halts and its final configuration $((s, t, u), i, q)$ satisfies $u \in D$. Then $f(x) = u$.

Definition 1.8. A *machine model* is a choice for (α, β, D, P).

Machine model	Space α	Time β	Domain D	Parameters P
Turing machines	ω	ω	ω	$\varnothing$
ITTMs	ω	On	2^ω	$\varnothing$
α-machines	α	α	α	$\varnothing$
OTMs	On	On	On	$\varnothing$
p-α-machines	α	α	α	α
p-OTMs	On	On	On	On

Table 2: Overview of the machine models discussed. α is admissible.

We now define machine models for ITTMs, OTMs, and α-machines. While for each machine type, the definition below differs from the one originally given in the literature (in terms of number of tapes and limit behavior), it is well-known that they are computationally equivalent.[2]

Definition 1.9. The model of *infinite time Turing machines* (ITTMs) by Kidder, Hamkins, and Lewis [6] is given by $(\omega, \mathsf{On}, 2^\omega, \varnothing)$.

We assume that the reader is familiar with the constructible hierarchy L and its relativizations $L[A]$ for sets A as discussed in [9, chapter 13] and [9, p. 192]. Write L_α for the αth level of L. In particular, $L = L_{\mathsf{On}}$.

Definition 1.10. Call $\alpha \leq \mathsf{On}$ *admissible* [18, section VII.1, prop. 1.5] if $(L_\alpha, \in)$ satisfies the replacement axiom for all Δ_0-formulas with parameters in L_α.

If α is admissible then $(L_\alpha, \in)$ is is closed under pairing, union, Σ_1-replacement, and Δ_1-separation, both with parameters in L_α [18, section VII.1].

Definition 1.11. For each admissible α, the model of α-machines is given by $(\alpha, \alpha, \alpha, \varnothing)$, where we identify α with the tapes that contain exactly one 1, so that $\delta < \alpha$ corresponds to the tape t with $t(\delta) = 1$.

The case $\alpha = \omega$ is known as *Turing machines* (TMs) [25], the case $\alpha = \mathsf{On}$ is a parameter-free version of Koepke's *ordinal Turing machines* (OTMs) [10], and the case $\alpha < \mathsf{On}$ is a parameter-free version of the α-machines introduced by Koepke and Seyfferth [12].

The models of parametrized α-machines (p-α-machines) are $(\alpha, \alpha, \alpha, \alpha)$. The case $\alpha = \mathsf{On}$ corresponds to the machines in [10] which we call p-OTMs, and the others correspond to those in [12], which we call p-α-machines.

The models are summarized in table 2. From now on, *any machine model discussed*, refers to any of these models. We shall also occasionally write M for any such machine model. In such contexts, M-machine refers to a machine for M and D_M, P_M to the domain and parameter space of M respectively.

[2]That is, they are equivalent up to simulation. See e.g. the remark in [7, section 3], [1, exercise 2.8.8] attributing [27], the remark at the end of [12, section 1], and appendix B.1.

1.2 Computable functions

As we can see M-machines as partial functions $D_M \to D_M$, the question arises for which partial functions there is a machine.

Definition 1.12. If m is an M-machine, we write $\{m\}$ for the partial function $D_M \to D_M$ induced by m. A partial function $f : D_M \to D_M$ is called *M-computable* (ITTM-computable, α-computable, p-α-computable, OTM-computable, p-OTM-computable) if there is an M-machine m with $\{m\} = f$.

In [12], p-α-machines are introduced as a machine model for α-recursion theory [18, part C]. A partial function is p-α-computable if and only if it is α-recursive [12, pp. 313, 315], that is, Σ_1-definable with parameters over L_α [18, p. 152]. Furthermore, a partial function is α-computable if and only if it is Σ_1-definable over L_α without parameters [12, pp. 313, 314].

Proposition 1.13 ([6, p. 570], [12, p. 315], [10, pp. 384, 385])**.** *For each of the machine models discussed, adding any finite number of tapes does not alter computability strength. (The class of computable functions is the same.)*

Adding extra input tapes, we can extend the definition of M-computable functions to multivariate functions.

Proposition 1.14 (Folklore)**.** *The following are α- and OTM-computable:*

- *ordinal addition, multiplication, and exponentiation (restricted to α);*
- *the Cantor pairing bijection $\mathsf{On}^2 \to \mathsf{On}$ (restricted to α^2) and its inverse.*[3]

There is also a bijective ITTM-computable pairing function $2^\omega \to 2^\omega$.

Proof. The first two items follow from [18, section VII.1.6], or, alternatively, by a machine-based proof similar to those in [10, section 3]. Reals x_0, x_1 can be coded in one real z by letting bit $2n + i$ of z be the nth bit of x_i. □

In general, we shall write $\langle \cdot, \cdot \rangle$ for all pairing functions, where it is clear from the context which is intended, and $A \otimes B$ for $\{\langle a, b \rangle : a \in A \wedge b \in B\}$.

Definition 1.15. The εth *column* of a class $B \subseteq \mathsf{On}$ and its εth *section* is given by $B^{[\varepsilon]} = \{\langle x, y \rangle \in B : x = \varepsilon\}$ and $B^{[<\varepsilon]} = \bigcup_{d<\varepsilon} B^{[\varepsilon]}$ respectively.

We think of M-machines as objects in D_M: we fix a suitable (effective) bijection between programs and ω—in the case of ITTMs we then presuppose a suitable injection of ω into 2^ω as well. Parameter-free machines can be thought of as natural numbers, and p-α-machines as elements of α. We also write $\{m\}$ for the partial function induced by the machine induced by the ordinal m.

[3]Also known as the Gödel pairing function. $\langle \delta, \varepsilon \rangle$ is defined as the order type of (δ, ε) under the well-order on pairs of ordinals obtained by first ordering on the maximum coordinate, then on the first coordinate, and finally on the second.

Theorem 1.16 (Simulation theorem [1, theorem 2.5.15], [18, p. 1.9]). *For each of the machine models discussed, there are universal machines that are able to simulate the computation of any other machine. That is, there is an ITTM m with $\{m\}(\langle n, x\rangle) = \{n\}(x)$ for any $n \in \omega$ and $x \in 2^\omega$, and similarly for the other machine models.*

Corollary 1.17. *For each model M discussed, The M-computable functions are closed under composition.*

Hidden in proofs of the above but good to mention, is that admissible α are closed under the operations of ordinal arithmetic (addition, multiplication, exponentiation), so the time bound plays no role when composing or simulating α-machines.

1.3 Semi-decidable degrees

If f is a partial function, we write $f(x)\downarrow$ (read: "$f(x)$ halts") if f is defined in x.

Definition 1.18. A class A is *M-decidable* if its characteristic function is M-computable. We write $\{m\} = A$, identifying classes with characteristic functions.

A class A is *M-semi-decidable* if it is the domain of some M-machine m, that is, if $x \in A \leftrightarrow \{m\}(x)\downarrow$ holds.

M-decidability implies M-semi-decidability. This cannot be reversed: the halting problem $H_M = \{\langle m, x\rangle : \{m\}(x)\downarrow\}$ is M-semi-decidable, as it is the domain of a universal machine (theorem 1.16), but not M-decidable. This can be shown by the well-known proof for Turing machines.

Theorem 1.19 (The halting problem [6, theorem 4.1]). *For any machine model M discussed, H_M is not M-decidable.*

Proof. If $\{m\}(\langle p, x\rangle)$ gives 1 if $\{p\}(x)\downarrow$ and 0 otherwise, consider the machine $\{m'\}(p')$ that halts if and only if $\{p'\}(p')$ does not. (This it can do by simulating $\{m\}(\langle p', p'\rangle)$, which will always halt and tell whether $\{p'\}(p')$ halts or not.) Then $\{m'\}(m')$ halts if and only if it does not. So m cannot exist. $\square$

Definition 1.20 (Turing). An *M-oracle-machine m* is an M-machine with an *oracle tape*, whose instructions may contain the *query state $q_?$*.

Its computations depend on m, its input, and an *oracle $A \subseteq D_M$*. When it is in state $q_?$ at step $\sigma + 1$, the oracle tape contains $100\ldots$ if there was an element of D written on the oracle tape at step σ that is an element of A, and otherwise the oracle tape contains just 0s. We write $\{m\}^A(x)$ for the result of the computation of m with oracle A and input x. The set Q of queries made in a computation $\{m\}^A(x)$ is split between the *positive* queries $Q \cap A$ and *negative* queries $Q \setminus A$.

Informally, oracle machines m are equipped with an oracle A, which, to the eyes of m, is decidable.

Definition 1.21. $A \subseteq D_M$ is said to *M-reduce* to $B \subseteq D_M$ if there is an M-oracle-machine m with $\{m\}^B = A$. (That is, there is an M-oracle-machine which, equipped with oracle B, is the characteristic function of A.) We write $A \leqslant_M B$.

If both $A \leqslant_M B$ and $B \leqslant_M A$ hold, then A and B are called *M-equivalent*, written $A \equiv_M B$. If $A \leqslant_M B$ and $A \not\equiv_M B$ hold, then we speak of strict M-reduction, written $A <_M B$.

Remark 1.22. In α-recursion theory, stricter notions of reducibility are considered, which Koepke and Seyfferth adopted for α-machines [12, p. 316]. We restrict ourselves to Turing's notion of reduction as defined above. Koepke and Seyfferth claim this is generally not transitive for p-α-machines [12, p. 316], referring to a similar result from metarecursion theory [3, theorem 1]. ⌟

Classes of M-equivalent sets are called *degrees*, and they are an object of study in computability theory. Since $\leqslant_{\mathrm{TM}}$ is transitive, we can lift it to the TM-degrees. For ITTMs, OTMs, and p-OTMs, $\leqslant_M$ is also transitive, and we do the same. Degrees that contain a semi-decidable are called *semi-decidable degrees*.

Since semi-decidable classes are defined as domains of machines, the following can be concluded.

Proposition 1.23 (Folklore)**.** *All M-semi-decidable classes M-reduce to H_M.*

1.4 ITTM-computability

Definition 1.24 (Hamkins and Lewis [6, pp. 573, 580])**.** Let x be a real. It is *writable* if there is an ITTM m with $\{m\}(0) = x$, *eventually writable* if there is an ITTM so that in the computation $\{m\}(0)$ there is a point in time after which the content of the output tape is always x, and *accidentally writable* if at any point in time x is written on the output tape in the computation $\{m\}(0)$.

For eventual and accidental writability, these computations need not halt.

Definition 1.25 (Hamkins and Lewis [6, pp. 573, 580])**.** If R is a partial order on a subset of ω, a real x is a code for R if for all $n, m \in \omega$ it is the case that the $\langle n, m\rangle$th bit of x is 1 if and only if $(n, m) \in R$.

A real x is a code for α if x codes a well-order of order type α. If there are writable, eventually writable, or accidentally writable codes for α, then α is called a writable, eventually writable, or accidentally writable ordinal respectively.

Proposition 1.26 (Hamkins and Lewis [6, theorem 2.2])**.** *The set* WO *of reals that code ordinals is ITTM-decidable.*

We shall use various properties of ordinal codes. In particular, given a codes x, y for α, β, an ITTM can

- remove minimal elements from x until exhausted ("clocking α" or "counting through α" [6, p. 572]), which takes exactly α removals [6, p. 572];

- thus compare α and β;
- compute codes for all $\delta < \alpha$ [6, theorem 3.7]; and
- code sequences $\{r_\delta < \delta < \alpha\}$ of reals in one real, called "coding along x" [6, p. 585].

An ITTM can iterate over codes y for $\beta < \alpha$ in ordinal order, as proved in appendix A.1. ITTMs can also do arithmetic on codes. Specifically, there is an ITTM which, given codes for δ, ε, produces a code for the Cantor pairing $\langle \delta, \varepsilon \rangle$. This is proved in appendix A.3.

Definition 1.27 (Hamkins and Lewis [6, p. 576]). An ordinal α is *clockable* if there is an ITTM m so that $\{m\}(0)$ halts in exactly α steps.

Write Won for the set of writable ordinals, Clk for the set of clockable ordinals, and Γ for the supremum of Won (which exists, since order types of well-orders on ω are necessarily countable). We shall use that Clk is decidable and Won is semi-decidable (by simulation), that $\mathsf{Won} = \Gamma$ [6, theorem 3.7], that $\mathsf{Clk} \neq \Gamma$ [6, theorem 3.5], that Clk and Won have the same order type [6, theorem 3.8], and that $\sup \mathsf{Clk} = \sup \mathsf{Won}$ [26, corolary 2.3].

Proposition 1.28 (Hamkins and Lewis [6, theorem 4.1]). *The weak halting problem $h = \{m < \omega : \{m\}(0)\downarrow\}$ is ITTM-semi-decidable and not ITTM-decidable.*

Write Wrr for the set of writable reals.

Proposition 1.29. *If $A \subseteq \mathsf{Wrr}$ is semi-decidable, then $A \leqslant_{\text{ITTM}} h$.*

Proof. Let $A = \operatorname{dom} \{m\}$. There is an ITTM $\{t\}$ which, given a machine n produces a machine e with $\{e\}(0) = \{m\}(\{n\}(0))$ by composing the instructions.

A can thus be reduced to h as follows: on input x, iterate over all $n < \omega$ and query $n \in h$. If $n \in h$, then simulate $\{n\}(0)$ and check if $\{n\}(0) = x$. If an n with $\{n\}(0) = x$ is found, then accept x if $\{t\}(n) \in h$ and otherwise reject. If no such n is found, reject x as well (because it is not writable). $\square$

Definition 1.30. $A \subseteq \omega$ is *recursively enumerable* (RE) if it is empty or the range of a total TM-computable function, and *generable* (GB) if there is a TM e with $A = \{x : 1x00\ldots$ is on the output tape at any point in the computation $\{e\}(0)\}$.

Write SD for TM-semi-decidability. ITTM-variants iRE, iSD, and iGB are defined analogously. In particular, for iGB we now consider sets of the form $\{x \in 2^\omega : 1x$ is on the output tape at any point in the computation $\{e\}(0)\}$.

Proposition 1.31 (Folklore). *A set is semi-decidable if and only if it is generable, if and only if it is recursively enumerable.*

This does not translate to ITTMs.

Proposition 1.32. *As in table 3, we have*

$\nearrow$	iRE	iSD	iGB
iRE	✓	X	X
iSD	X[4]	✓	X
iGB	✓	✓	✓
ITTM-decidable	X	✓	X

Table 3: Implication table for definitions of enumerability for ITTMs.

1. iGB $\to$ iRE $\wedge$ iSD,
2. iRE $\wedge$ iSD $\not\to$ iGB,
3. iSD $\not\to$ iRE,
4. iRE $\not\to$ iSD,
5. *and ITTM-decidable implies* iSD, *but not* iRE *or* iGB.

Proof. 1. As shown in [6, p. 570], ITTM-computations either halt or start to repeat in ω_1 steps. Hence, if $A \neq \emptyset$ is generated by ITTM m, then there is an ITTM whose range is A: on ordinal input α, it simulates m until a real is generated, then for another α steps, and outputs the last real generated. On non-ordinal inputs, it outputs the first real generated by m.

For iGB $\to$ iSD, consider an ITTM that, given a real x, simulates a generator indefinitely and halts only if $1x$ is ever on the output tape.

2. 2^ω is iRE and iSD but not iGB: it contains non-accidentally-writable reals (as there are only countably many accidentally writable reals), and by definition ITTM-generable sets contain only accidentally writable reals.

3. From the proof of [26, corollary 2.3] (more explicitly, [1, corollary 3.6.10]) it can be concluded that the set Naw of non-accidentally-writable reals is ITTM-decidable. Hence, it is iSD. It is not iRE, since nonempty iRE sets must contain writable reals. (If A is the range of an ITTM-computable function f, then $f(0)$ is a writable element of A.)[4]

4. In [6, theorem 5.1] it is shown that h strictly reduces to $h^\triangledown = \{e : \{e\}^h(0)\downarrow\}$, hence $h^\triangledown$ is not iSD. To see that it is iRE, we recall that Naw is ITTM-decidable and so $Z = \mathsf{WO} \cap \mathsf{Naw}$ is too. From the proof of [26, corollary 2.3] and the observation that all computations on writable inputs either halt in less than Γ steps or diverge (formalized in proposition 1.36), it follows that h is ITTM-computable from any ordinal in Z. So the function

$$x \mapsto \begin{cases} e & \text{if } x \text{ codes } \lambda + e \text{ in } Z \text{ and } \{e\}^h(0) \text{ halts in } \lambda \text{ steps;} \\ \min h^\triangledown & \text{otherwise} \end{cases}$$

[4] If an iSD set $A = \mathrm{dom}\{e\}$ contains a writable real a, then it is iRE: it is the range of the ITTM-computable function

$$f(\langle x, y\rangle) = \begin{cases} x & \text{if } y \text{ codes an ordinal } \alpha \text{, and } \{e\}(x) \text{ halts in } \alpha \text{ steps;} \\ a & \text{otherwise.} \end{cases}$$

is ITTM-computable and has range $h^\triangledown$ (since all halting ITTM-computations are countable [6, theorem 1.1]).

5. ITTM-decidabality implies ITTM-semi-decidability. In (3) we find an ITTM-decidable set that is not iRE, and in (2) one that is not iGB. □

Given $\alpha \in \mathsf{Won}$, order the machines m writing it first by the runtime of $\{m\}(0)$ and then by machine code. If e is the least under this ordering, then $\{e\}(0)$ is the *canonical code* of α. Given any code for $\alpha \in \mathsf{Won}$, an ITTM can compute its canonical code by simultaneously simulating all $\{m\}(0)$. We often identify Won with the set of canonical codes, and act likewise for Clk.

Proposition 1.33. *The set* Wrr *of writable reals is ITTM-generable.*

Proof. There is an ITTM which simultaneously simulates all computations $\{m\}(0)$ and shows the output of halting computations on the output tape. □

The following proposition will be used frequently in the thesis. A proof—which is more technical than insightful—can be found in appendix A.1.

Proposition 1.34. *The set of writable ordinals* Won *(here seen as the set of canonical codes for writable ordinals) is ITTM-generable in ordinal order.*

Proposition 1.32 sheds light on why priority arguments do not translate well to ITTM-computability. In the same vein, we note:

Proposition 1.35. *There is generally no ITTM-computable well-order on the reals so that, given $x \in 2^\omega$, an ITTM can generate its predecessors and halt.*

Proof. Assume that such a well-order $\prec$ existed. By assumption, for every $x \in 2^\omega$, an ITTM can generate $\mathrm{pred}(x) = \{y : y \prec x\}$ and halt. All halting ITTM-computations are countable [6, theorem 1.1], so for every $x \in 2^\omega$, the cardinality of $\mathrm{pred}(x)$ is countable. It follows that the order type of $(2^\omega, \prec)$ is at most ω_1, as this is the largest order type in which all predecessor sets are countable. As 2^ω is uncountable, the order type of $(2^\omega, \prec)$ cannot be countable. Hence the order type is ω_1, and so 2^ω is in bijection with ω_1, from which $2^{\aleph_0} = \aleph_1$ follows. So the existence of $\prec$ implies the continuum hypothesis, which is a statement independent from ZFC [9, chapter 14]. □

When restricted to Won, halting ITTM-computations have writable length.

Proposition 1.36. *If $w \in \mathsf{Wrr}$ and the ITTM-computation $\{e\}(w)$ halts, then it halts in clockable time.*

Proof. Let $w \in \mathsf{Wrr}$ and assume $\{e\}(w)\downarrow$. Modify the program of e to obtain an ITTM e' so that, on input 0, e' first writes a code for w on the input tape, and then acts as e normally would.

More formally: let m be a machine that writes w. Let Q_e be the states contained in the instructions of e, and let $f : Q_e \to Q'_e$ be a bijection so that Q'_e contains no states mentioned in the instructions of m. Let q be the final configuration in the computation $\{m\}(0)$, and i the final cell position, and s the final symbol on cell i. The instructions for e' are then given by the instructions of m, the instructions obtained by applying f to all states in instructions of e, together with instructions that, starting from state q, move left i times and then enter state $f(q)$.

Then $\{e'\}(0) = \{e\}(w)$. Let σ, τ be their respecting halting times. By definition of e', we have $\sigma \geq \tau$, and by definition of clockability we have $\sigma \in \mathsf{Clk}$. □

Generally speaking, when restricted to Won, ITTMs and p-Γ-machines are somewhat similar, as outlined in appendix B.2. In particular, a $A \subseteq \Gamma$ is p-Γ-semi-decidable if and only if there is an ITTM-semi-decidable $A' \subseteq 2^\omega$ such that, up to coding, $A' \cap \mathsf{Won} = A$. They are not the same, however. In particular, the set of codes of writable ordinals is not ITTM-decidable (appendix A.2), and so p-Γ-reductions do not directly translate to ITTM-reductions on subsets of Won (there is no general way to reject reals that do not code ordinals in Γ).

1.5 Definability

Transfinite computability and the constructible hierarchy are closely related. In [12, lemma 3], Koepke and Seyfferth show that, if δ is a limit ordinal, M-computations of length up to δ are uniformly Δ_1-definable (with parameters) over $L_\delta[t]$, where t is the initial tape of the computation (containing the input and parameter). For every admissible $\alpha \leq \mathsf{On}$, there is an α-machine effectively enumerating L_α [18, section VII.1.7], in the sense that, given codes for sets in L_α, the answer to basic set-theoretic questions such as membership can be computed. Both can be relativized to oracles A, replacing L with $L_\delta[A]$ [12, pp. 313, 314]. Similarly, Hamkins and Lewis have shown that ITTMs can, given a code for an ordinal δ, effectively iterate over L_δ [6, pp. 585, 586], and thus generate L_Γ.

1.6 Bounded simulation and approximations

Definition 1.37. Given M-machine m, define the δ-step approximation

$$\{m\}_\delta(x) = \begin{cases} \{m\}(x) & \text{if the computation } \{m\}(x) \text{ takes strictly less than } \delta \text{ steps} \\ \uparrow & \text{otherwise} \end{cases}$$

where $\uparrow$ means that x is outside the domain of $\{m\}_\delta$. Define $\{m\}^X_\delta(x)$ similarly.

Proposition 1.38. *For the machine models M discussed, if δ is strictly below the time bound, $\{m\}_\delta(x)$ is M-computable in $\langle \delta, x\rangle$, in the strong sense that there is also an M-computable function $H(\delta, x)$ that computes whether $\{m\}_\delta(x)\downarrow$.*[5]

Proof. For ITTMs, this can be done by bounding the runtime of a universal machine as in [6, theorem 4.5]. For α-machines, this follows from a similar proof, or alternatively the results in [12, pp. 313–314]. $\square$

Corollary 1.39. *$\{m\}^A_\delta(x)$ is similarly strongly M-computable in the oracle A. In particular, if A is M-decidable, then $\{m\}^A_\delta(x)$ is M-computable.*

Proof. For ITTMs, this follows follows from the proof of proposition 1.38, where the simulating machine forwards queries made to A. For α-machines, this follows from the results in [12, pp. 313–314]. $\square$

Definition 1.40. If $C \subseteq \mathsf{On}$ is p-OTM-semi-decidable, define the *δ-step simulation* $C_\delta = \{\beta < \delta : \{e\}_\delta(\beta)\downarrow\}$, where e is an implicitly fixed p-OTM with $\operatorname{dom}\{e\} = C$.

If $C \subseteq \mathsf{Won}$ is ITTM-semi-decidable, likewise define the approximation $C_\delta = \{\beta \in \mathsf{Won} : \beta < \delta : \{e\}_\delta(\beta)\downarrow\}$.

As a direct corollary of proposition 1.38, we find:

Proposition 1.41. *If C is p-OTM-semi-decidable, there is a p-OTM m so that*

$$\{m\}(\delta, \varepsilon) = \begin{cases} 1 & \textit{if } \varepsilon \in C_\delta \\ 0 & \textit{otherwise} \end{cases}$$

and similarly for OTMs and ITTMs.

Proposition 1.42. *If $C \subseteq \mathsf{On}$ is p-OTM-semi-decidable, then $C = \bigcup_{\delta\in\mathsf{On}} C$, and similarly for OTMs.*

If $C \subseteq \mathsf{Won}$ is ITTM-semi-decidable, then $C = \bigcup_{\delta\in\mathsf{Clk}} C_\delta$.

Proof. All halting computations necessarily have ordinal length, so for p-OTMs and OTMs this is immediate by definition of C_δ. The result for ITTMs follows from proposition 1.36. $\square$

[5]For ITTMs, δ is represented by a real coding it, as in definition 1.25.

Chapter 2

The Friedberg-Muchnik theorem

The Friedberg-Muchnik theorem for M states that there are M-semi-decidable A, B so that $A \not\leq_M B$ and $B \not\leq_M A$. A and B are defined in terms of decidable approximations, for which bounded simulation is used extensively.

We first discuss a proof for Turing machines, and the generalizations to ITTMs due to Hamkins and Lewis. We then give a new proof for the known generalization to OTMs, and sketch proofs for α-machines and p-OTMs.

2.1 For Turing machines

We present a proof based on that of [24, theorem 6.2.1]. Roughly, the idea is to generate approximations $A_0 \subseteq A_1 \subseteq \ldots$ and $B_0 \subseteq B_1 \subseteq \ldots$ that are increasingly incomparable, so that the respective unions A and B are incomparable. If the approximation steps can be carried out by a Turing machine, then A and B are semi-decidable.

In order for A and B to be incomparable, it is sufficient to satisfy the requirements

$$\{e\}^A \neq B \tag{R_e^A}$$

and

$$\{e\}^B \neq A \tag{R_e^B}$$

for every $e < \omega$. Let $\{R_i : i \in \omega\}$ be a computable well-ordering of these requirements. Requirement R_i has higher *priority* than R_j if $i < j$.

The sets A_s and B_s are defined recursively. Let X, Y be A, B or B, A. The idea is to find for each requirement (R_e^X) a w witnessing $\{e\}^X \neq Y$, so with either $\{e\}^X(w)\uparrow$—in which case (R_e^X) holds because $\{e\}^X$ is not total—$\{e\}^X(w) = 1$ while $x \notin Y$, or $\{e\}^X(w) = 0$ while $w \in Y$. This is done by keeping an initial guess w, called the witness attempt, computing increasing approximations $\{e\}_s^{X_s}(w)$ of $\{e\}^X(w)$, and if it turns out that $\{e\}_s^{X_s}(w) = 0$, adding w to Y_s. However, this might change the outcome of another computation $\{e'\}_s^{Y_s}(w')$. Thus adding w to Y in order to satisfy (R_e^X) might *injure* a requirement $(R_{e'}^Y)$.

To combat this, once a computation $\{e\}_s^{X_s}(w) = 0$ is found and w is added to Y, a *restraint* r is recorded, which is larger than the supremum of the queries made in $\{e\}_s^{X_s}(w) = 0$. If no elements below r are ever added to X, then $\{e\}^X(w) = \{e\}_s^{X_s}(w)$, the computation $\{e\}_s^{X_s}(w) = 0$ is preserved and (R_e^X) is satisfied.

To avoid a situation where the restraints of some requirements might make it impossible to satisfy others, we allow requirement R_i to injure all requirements $R_j, j > i$: if R_i adds any witness to X or Y those requirements have their witness attempt pushed outside of R_i's new restraint. Now for every requirement there are only finitely many requirements that can injure it.

2.1.1 The approximation algorithm

We now formally describe the approximation steps of A and B. See algorithm 1 for details. At step s, initially let A_s and B_s be the unions of their earlier approximations (initially, $A_0 = B_0 = \varnothing$), and let the $w_{i,s}, r_{i_s}, i < s$ be the current witness candidate and restraint for requirement R_i (either $w_{i,s-1}, r_{i-1,s-1}$ or $0, 0$ in case $s = 0$). Then consider requirements $R_0, \dots, R_{s-1}$. If R_i is of the form (R_e^A), then check whether $\{e\}_s^{A_s}(w_{i,s}) = 0$. If so, and if $w_{i,s} \notin B_s$ holds, then R_i *receives attention*: $w_{i,s}$ is added to B_s, and the restraint $r_{i,s}$ is set to the smallest number above all queries made in the computation.

If R_i received attention, the requirements $R_{i+1}, \dots, R_{s-1}$ are *injured*, and their witness attempts are pushed out of $r_{i,s}$ so that they cannot injure R_i in the future. Specifically, the $w_{j,s}$ of the injured requirements are pushed above all witnesses $w_{k,s}$ and restraints $r_{k,s}$ of requirements R_k of higher priority (with $k < j$).

If R_i is of the form (R_e^B), act similarly, reversing the roles of A and B. The final task in step s is to, because R_s will be considered for the first time in step $s + 1$, set an initial witness candidate $w_{s,s}$ above all $w_{i,s}, r_{i,s}, s < i$.

We require witnesses of R_i to be of the form $\langle i, v\rangle$, so that every $w < \omega$ can only be a witness candidate for one requirement.

It remains to show that $A = A_{<\omega} = \cup_{s<\omega} A_s$ and $B = B_{<\omega}$ are as desired.

2.1.2 Correctness

To see that A and, similarly, B, is semi-decidable, note that if $x \in A$ then it enters A at some step s. Hence A is the domain of the machine that continuously refines its approximation of A and halts only if its input is added to some A_s.

The treatment of R_e^A and R_e^B are symmetrical. Let X, Y be A, B or B, A. We note an immediate fact about algorithm 1.

Lemma 2.1 (Attention lemma). *Let $R_i = \{e\}^X$. If $\{e\}_s^{X_s}(w_{i,s}) = 0$ and $w_{i,s} \notin Y$, then R_i either receives attention or is injured in some stage $t \geq s$.*

Algorithm 1: Step s of the Friedberg-Muchnik algorithm.

```
1: $A_s \leftarrow A_{<s}$;   $B_s \leftarrow B_{<s}$
2: **for** $i < s$ **do**
3:     $w_{i,s} \leftarrow \lim_{t<s} w_{i,t}$;   $r_{i,s} \leftarrow \lim_{t<s} r_{i,t}$
4: **end for**
5: **for** $i < s$ **do**
6:     **let** $e, X, Y$ such that $R_i = (R_e^X)$ and $X \neq Y$
7:     **if** $\{e\}_s^{X_s}(w_{i,s}) = 0 \wedge w_{i,s} \notin Y_s$ **then**          $\triangleright$ $R_i$ *receives attention*
8:         $Y_s \leftarrow Y_s \cup \{w_{i,s}\}$
9:         $r_{i,s} \leftarrow \sup\{x : x$ is queried in $\{e\}_s^{X_s}(w_{i,s}) = 0\}$
10:        **for** $j \in (i,s)$ **do**                                  $\triangleright$ $R_j$ is *injured*
11:            $w_{j,s} \leftarrow \min\{w \in \{j\} \otimes \omega : (\forall k \leq j)\ w > w_{k,s}, r_{k,s}\}$
12:            $r_{j,s} \leftarrow 0$
13:        **end for**
14:    **end if**
15: **end for**
16: $w_{s,s} \leftarrow \min\{w \in \{s\} \otimes \omega : (\forall k < s)\ w > w_{k,s}, r_{k,s}\}$
```

The following would go unmentioned in recursion theory, but its transfinite analogies will be less obvious. Hamkins and Lewis named it the reflection lemma [5] in their treatment of the Friedberg-Muchnik theorem for ITTMs.

Lemma 2.2 (Reflection lemma). *If $\{e\}^X(x) = 0$, there are cofinally many $s < \omega$ with $\{e\}_s^{X_s}(x) = 0$.*

Proof. Consider the running time t of the computation $\{e\}^X(x)$, and the set of queries Q made within. Because t is finite (the time bound of TMs is ω), Q is also finite, and hence there is a state $S \geq t$ at which all elements of Q have already entered X_S. (For each $q \in Q$ there must be an s^q with $q \in X_{s^q}$. Q is finite so the supremum of the s^q is finite as well.) Then $\{e\}^X(x) = \{e\}_s^{X_s}(x)$ for all $s \geq S$. □

Secondly, we note, by considering priority, that requirements eventually *settle* during the approximations.

Lemma 2.3 (Finite injury). *Each requirement is injured at most finitely often.*

Proof. This follows by induction from two observations:

- Requirement R_i is injured only when an R_j, $j < i$ receives attention.
- Once a requirement receives attention, it must be injured before it can receive attention again. (Hence, if a requirement is injured finitely often, it also receives attention only finitely often.)

So if the finitely many requirements R_j, $j < i$ are injured finitely often, they receive attention finitely often, and then R_i is injured finitely often. □

Corollary 2.4 (Settling lemma). *For each requirement R_i, $w_i = \lim_{t\to\infty} w_{i,t}$ and $r_i = \lim_{t\to\infty} r_{i,t}$ exist. There is a (first) stage s in which requirement R_i* settles*: $w_{i,s} = w_i$, $r_{i,s} = r_i$, and R_i is never injured or receives attention after stage s.*

If a requirement settles, it is either injured and never receives attention after (*injured last*), or it receives attention and is never injured after (*receives attention last*).

Lemma 2.5 (Preservation lemma). *Assume $R_i = R_e^X$ settles. If it is injured last, then $w_i \notin A \cup B$. If it receives attention last, then $\{e\}^X(w_i) = 0$ and $w_i \in Y$.*

Proof. Since requirement R_i only takes witness candidates in $\{i\} \otimes \omega$, every x is only ever a witness candidate for at most one requirement. Furthermore, x is only added to A or B when a requirement that has it as its witness candidate receives attention. So if R_i is injured last, it had gotten w_i as a new witness candidate then (since witness candidates are increasing over injuries, cf. line 11 of algorithm 1), and so it never received attention while w_i was its witness candidate. It follows that $w_i \notin A \cup B$.

If R_i receives attention last, say in stage s, then $w_{i,s} = w_i$ and $w_{i,s} \in Y_s$. Furthermore, $\{e\}_s^{X_s}(w_i) = \{e\}_s^{X_s}(w_{i,s}) = 0$ and $r_i = r_{i,s}$ lies above the queries made in the computation. Requirements R_j, $j < i$ never receive attention after stage s since that would injure R_i, so no elements are added to X by merit of some R_j, $j < i$. After stage s, witness candidates for requirements R_j, $j > i$ always lie above r_j (by line 11), and so no element below r_i is added by merit of the R_j, $j > i$. Hence, the computation is preserved: $(A \cup B) \cap (r_i + 1) = (A_s \cup B_s) \cap (r_{i,s} + 1)$ holds, thus $\{e\}^X(w_i) = \{e\}_s^{X_s}(w_i) = 0$. □

The proof of the Friedberg-Muchnik theorem rests solely on the settling, reflection, preservation, and attention lemmas.

Proof of the Friedberg-Muchnik theorem for TMs. Let (R_e^X) be a requirement. By the settling lemma, it settles at some stage s. It was either last injured or it last received attention. If it last received attention, then, by the preservation lemma, there is a $w \in Y$ with $\{e\}^X(w) = 0$, and so (R_e^X) is satisfied.

Assume on the other hand that (R_e^X) was last injured. Let w be its final witness candidate. By the preservation lemma we have $w \notin Y$. Assume towards a contradiction that (R_e^X) is not satisfied. Then $\{e\}^X(w) = 0$ since $w \notin Y$. By the reflection lemma, there is a $t > s$ with $\{e\}_t^{X_t}(w) = 0$. By the attention lemma, (R_e^X) receives attention in some stage $t' \geq t$, contradicting that it had settled. □

2.2 For ITTMs

There are several challenges in transforming the classical Friedberg-Muchnik proof to the system of ITTMs. In light of proposition 1.35 and the fact that 2^ω is not ITTM-generable, it difficult to define sets by a union of increasing approximations. Furthermore, while halting TM-computations make at most a finite number of oracle queries, ITTMs can make infinitely many, making it harder to store restraints.

In [5, section 4], Hamkins and Lewis circumvent these issues by doing the Friedberg-Muchnik construction in the writable reals, using Clk-many stages.

2.2.1 Strategy

Hamkins and Lewis employed a similar strategy as in section 2.1. As there are countably many ITTMs, the requirements are again

$$\{e\}^A \neq B \tag{R_e^A}$$

and

$$\{e\}^B \neq A \tag{R_e^B}$$

for every $e < \omega$, where of course $\{e\}$ refers to ITTM e, and A, B are sets of reals. Specifically, A and B will only contain writable reals, making the search for and storage of witnesses feasible: an ITTM can enumerate the writable reals by simulating all computations $\{e\}(0)$, and store a writable real in the form of an ITTM that writes it.

The incomparable A and B are defined by an algorithm similar to algorithm 1. A step is executed for each clockable ordinal α. Restraints are coded not as upper bounds of queries but as sets of queries, represented by a computation: if the negative queries in $\{e\}_\alpha^{A_\alpha}(w) = 0$ should not be added to A later, this restraint can be finitely coded by storing e, a code for α, and a code for w. From this, A_α and the computation can be recovered, and hence the set of queries.

Another difference is in the assignment of new witnesses. If R_i is injured at stage α, its witness $w_{i,\alpha}$ should be pushed to a real outside of all past and present witnesses, and restraints of higher priority. In algorithm 1, this is achieved by taking a supremum over these values, but since the writable reals are not as nicely ordered, an alternative is needed.

Lemma 2.6 (Hamkins and Lewis). *For every clockable ordinal α, there is a writable real x that does not appear accidentally in any computation $\{e\}_\alpha(0)$.*

Proof. There are ω many ITTMs e, and each computation $\{e\}_\alpha(0)$ has countable length. So by Cantor's diagonal argument, there are reals that do not appear accidentally in any of these computations. The key observation is that there is an ITTM which, on input 0, carries out this diagonalization: it simulates the computations, producing a real x such that the $\langle e, i\rangle$th bit of x is not the $\langle e, i\rangle$th bit of the real written on the output tape in the computation $\{e\}_\alpha(0)$ at step β, where i is the βth element in the ordering of ω induced by a fixed code of α. $\square$

The changes are summarized in algorithm 2. In every step, all ω requirements are considered. The initial step consists of setting $w_{i,0} = i$ for all i. Approximations of X and Y can be stored on a single tape each, by coding its elements (all writable reals) as ITTMs producing them.

A note on how to read these algorithms: we often gloss over the fact that limit steps exist. For compound limits especially, this can be problematic. Parts of the tape used for intermediary values will be scrambled in such steps. Luckily, M-machines may detect that they are in a compound limit step by flashing a cell on every limit step, as in example 1.6, and so can can often be made to recover.

Algorithm 2: Step $\alpha > 0$ of the Friedberg-Muchnik algorithm for ITTMs.

```
 1: let $w_{i,\alpha}$ and $r_{i,\alpha}$ be their previous value (or lim inf as per machine behavior)
 2: for $i < \omega$ do
 3:     let $e, X, Y$ such that $R_i = (R_e^X)$ and $X \neq Y$
 4:     if $\{e\}_s^{X_\alpha}(w_{i,\alpha}) = 0 \wedge w_{i,\alpha} \notin Y_\alpha$ then        ▷ $R_i$ receives attention
 5:         $Y_\alpha \leftarrow Y_\alpha \cup \{w_{i,\alpha}\}$
 6:         $r_{i,\alpha} \leftarrow \{x : x$ is queried negatively in $\{e\}_s^{X_s}(w_{i,s}) = 0\}$
 7:         for $j \in (i, \omega)$ do                                   ▷ $R_j$ is injured
 8:             $w_{j,\alpha}$ gets a fresh unique real outside higher restraints (lemma 2.6)
 9:             $r_{j,\alpha} \leftarrow \emptyset$
10:         end for
11:     end if
12: end for
```

For this particular algorithm, it will be proved below that each requirement is injured but finitely often, and hence all stored values are unaffected by limit behavior—provided that care is taken in how they are stored. For example, the ITTM could divide a tape in ω^3 subtapes, and use a separate subtape for each variable. All that has to be done in (compound) limit steps is to clear some working space.

Remark 2.7. While describing Turing machines in terms of line-based algorithms is common, one might wonder how a transfinite M-machine determines at limit steps which line of an algorithm it was working on. This can be achieved by having a "line-tape" on which the current algorithm line n is coded as n 1s followed by zeroes. In limit steps, this contains the lim inf of the lines it has been working on thus far. If the line number converged up to that step, it can continue working on this line as expected. If the line number did not converge, then it was working on a finite number of lines in, say, a loop. The contents of the line-tape then points to the first line of the loop, where the machine can choose to exit or continue the loop based on the algorithm. See [10] for a more exhaustive discussion of how these algorithms can be interpreted. ⌟

By design of the algorithm (combined with proposition 1.36, and corollary 1.39), we note the following.

Lemma 2.8 (Attention lemma). *Let $R_i = \{e\}^X$ and $s \in \mathsf{Clk}$ If $\{e\}_s^{X_s}(w_{i,s}) = 0$ and $w_{i,s} \notin Y$, then R_i either receives attention or is injured in some stage $t \geq s$.*

2.2.2 Correctness

Define $A = \bigcup_{\alpha \in \mathsf{Clk}} A_\alpha$ and $B = \bigcup_{\alpha \in \mathsf{Clk}} B_\alpha$, where the approximations are defined by the algorithm discussed above. Proving that A and B are incomparable goes along the same lines as for TMs, but the proof of the reflection lemma in particular uses an interesting trick.

Lemma 2.9 (Reflection lemma [5, p. 521]). *Let X be A or B. If w is writable and $\{e\}^X(w) = 0$, there are cofinally many $\alpha \in \mathsf{Clk}$ with $\{e\}_\alpha^{X_\alpha}(w) = 0$.*

Proof. Because steps are only executed for clockable ordinals, there is an ITTM that continuously refines approximations of A and B. (It can do this by simulating all computations $\{e\}(0)$ and executing a step whenever one halts.) Eventually, after it has done Clk steps, the final contents of A and B are on the tape of the machine. (The ITTM cannot recognize this, it will continue to search for new clockables, without success, and never halt.)

Now modify this ITTM to, after executing at least $\beta \in \mathsf{Clk}$ stages, halt as soon as it finds $\{e\}^{X'}(w) = 0$ for its current approximation X' of X. The machine first writes a code for β (recall, clockable ordinals are writable), and then takes out a minimal element every time it has executed a stage until the well-order is exhausted. Once the well-order is exhausted, it continues refining its approximation X', and now checks after every stage whether $\{e\}^{X'}(w) = 0$. By the assumption $\{e\}^X(w) = 0$ and the above, the ITTM must halt, since eventually $X' = X$ holds. Its halting time is by definition a clockable ordinal (as it ignores its input), and so $\{e\}^{X'}(w) = 0$ must hold for some X' in $\{X_\alpha : \beta \geq \alpha \in \mathsf{Clk}\}$. □

By the exact same reasoning as in the proof of lemma 2.3, we have

Lemma 2.10 (Finite injury). *Each requirement is injured at most finitely often.*

Corollary 2.11 (Settling lemma). *For each requirement there is a stage $\alpha \in \mathsf{Clk}$ in which it* settles*: it is never injured or receives attention in stages $\beta \geq \alpha$.*

Proof. By the above, we know that each requirement receives attention finitely often. So requirement R_i has settled after the finitely many stages in which it is injured or receives attention. It is left to show that no finite sequence $\alpha_1, \dots, \alpha_n$ of clockable ordinals is cofinal in the clockable ordinals. Indeed: for each of the α_i there is a machine m_i so that $\{m_i\}(0)$ halts in α_i steps, and there is an ITTM which sequentially simulates the computations $\{m_i\}(0)$ and then halts, halting after a number of steps greater than any of the α_i. □

Lemma 2.12 (Preservation lemma). *Assume $R_i = R_e^X$ settles. If it is injured last, then $w_i \notin A \cup B$. If it receives attention last, then $\{e\}^X(w_i) = 0$ and $w_i \in Y$.*

Proof. Let α be the last stage in which R_i is either injured or receives attention.

Assume R_i is injured last in stage α. Because injured witnesses get fresh reals as a new witness candidates, a writable real is only ever a witness candidate for at most one requirement. So w_i is only ever a witness candidate for R_i, and (because it is fresh with respect to past witnesses as well) stage α is the first time that $w_{i,\alpha} = w_i$ holds. Witness candidates are only added to $A \cup B$ if its requirement receives attention, and, by assumption, R_i does not receive attention after getting w_i as its witness candidate. Hence $w_i \notin A \cup B$.

When a requirement receives attention in stage α, its restrains all negative queries made in the computation $\{e\}_\alpha^{X_\alpha}(w_{i,\alpha})$ and from that point on, the witness candidates of requirements of lower priority lie outside this restraint. Hence, if R_i requires attention last, say in stage α, no requirement $R_j, j > i$ will add elements of $r_i = r_{i,\alpha}$ to X. Furthermore, since requirements $R_j, j < i$ receiving attention implies R_i being injured, requirements of higher priority do not add elements of r_i to X either. It follows that $X_\alpha \cap r_i = X \cap r_i$, and so we find $\{e\}^X(w_i) = \{e\}_\alpha^{X_\alpha}(w_{i,\alpha}) = 0$. □

Having proved a reflection, preservation, settling, and attention lemma, a Friedberg-Muchnik theorem follows as on page 19.

Theorem 2.13 (Friedberg-Muchnik theorem for ITTMs, Hamkins and Lewis)**.** *There are ITTM-incomparable ITTM-semi-decidable sets (with only writable reals).*

On the whole, there are two novelties in this proof, compared to that for Turing machines: solving storage-related challenges using writability, and solving reflection by only adding witnesses in clockable stages.

2.3 For OTMs

In [7], Hamkins and Miller proved a Friedberg-Muchnik theorem for ordinal register machines (ORMs) by employing a similar strategy as for ITTMs above: introducing notions of clockability and writability, proving that the clockables have a supremum, and making sure to only execute stages on clockable times. Then reflection follows by a similar proof as for lemma 2.9. They then show that OTMs can simulate ORMs and vice versa.

We present a direct proof for OTMs that, instead of introducing clockability, uses the fact that there are only "set-many" requirements to prove reflection.

2.3.1 Strategy

The requirements are

$$\{e\}^A \neq B \qquad (R_e^A)$$

and

$$\{e\}^B \neq A \tag{R_e^B}$$

for every $e < \omega$, where $\{e\}$ refers to OTM e, and A, B are classes of ordinals. We go back to an algorithm in the style of algorithm 1. Because each requirement will be injured only finitely often, no special care has to be taken with storage in regards to limit behavior, apart from reserving separate space for each requirement. For example, the tape can be divided into $\omega \times 2$ subtapes, where at step α the value of $w_{i,\alpha}$ is stored on the $(i,0)$th subtape and $r_{i,\alpha}$ on subtape $(i,1)$. This is similar to the storage strategy used in [12, section 4] for a Friedberg-Muchnik theorem for p-α-machines. We also reserve two tapes t_A, t_B for storing A and B, so that at the start of stage α, the βth cell of t_A contains a 1 if and only if $\beta \in A_{<\alpha}$, and likewise for B. Since elements are never taken out of approximations, at the start of limit stages λ tapes t_A, t_B contain $A_{<\lambda}, B_{<\lambda}$ respectively.

It follows that the behavior described in algorithm 3, which is very similar to algorithm 1, can be expressed by an OTM. For step 0, set $w_{i,0} = \langle i, 0\rangle$ and $r_{i,0} = 0$. In particular, we note—again letting X, Y be either A, B or B, A:

Lemma 2.14 (Attention lemma). *Let $R_i = \{e\}^X$ and $\alpha \in \mathsf{On}$. If $\{e\}_\alpha^{X_\alpha}(w_{i,\alpha}) = 0$ and $w_{i,\alpha} \notin Y$, then R_i either receives attention or is injured in some stage $\beta \geq \alpha$.*

Algorithm 3: Step $\alpha > 0$ of the Friedberg-Muchnik algorithm for OTMs.

```
1: $A_\alpha \leftarrow A_{<\alpha}$;   $B_\alpha \leftarrow B_{<\alpha}$
2: for $i < \alpha$ do
3:     $w_{i,\alpha} \leftarrow \lim_{t<\alpha} w_{i,t}$;   $r_{i,\alpha} \leftarrow \lim_{t<\alpha} r_{i,t}$
4: end for
5: for $i < \omega$ do
6:     let $e, X, Y$ such that $R_i = (R_e^X)$ and $X \neq Y$
7:     if $\{e\}_\alpha^{X_\alpha}(w_{i,\alpha}) = 0 \land w_{i,\alpha} \notin Y_\alpha$ then        ▷ $R_i$ receives attention
8:         $Y_\alpha \leftarrow Y_\alpha \cup \{w_{i,\alpha}\}$
9:         $r_{i,\alpha} \leftarrow \sup\{x : x$ is queried in $\{e\}_\alpha^{X_\alpha}(w_{i,\alpha}) = 0\}$
10:        for $j \in (i, \omega)$ do                                     ▷ $R_j$ is injured
11:            $w_{j,\alpha} \leftarrow \min\{w \in \{j\} \otimes \mathsf{On} : (\forall k < j)\, w > w_{k,\alpha}, r_{k,\alpha}\}$
12:            $r_{j,\alpha} \leftarrow 0$
13:        end for
14:    end if
15: end for
```

2.3.2 Correctness

Again, the proof of lemma 2.3 gives

Lemma 2.15 (Finite injury). *Each requirement is injured at most finitely often.*

Corollary 2.16 (Settling lemma)**.** *All requirements settle: for each requirement R_i, there is a stage α so that R_i is neither injured nor receives attention in a later stage, and the limits $w_i = w_{i,\alpha} = \lim_{\alpha \in \mathrm{On}} w_{i,\alpha}$ and $r_i = r_{i,\alpha} = \lim_{\alpha \in \mathrm{On}} r_{i,\alpha}$ exist.*

Proof. Let $i < \omega$. By the finite injury lemma, R_i is injured at most finitely often, so there is a stage α' after which R_i is never injured. Requirements do not receive attention twice without being injured in-between, so there is a stage $\alpha \geq \alpha'$ after which R_i is never injured and never receives attention. It follows that for all $\beta > \alpha$ we have $w_{i,\beta} = w_{i,\alpha}$ (since R_i only gets a new witness candidate when it is injured) and $r_{i,\beta} = r_{i,\alpha}$ (since the restraint only changes when R_i receives attention), and hence $w_{i,\alpha} = \lim_{\beta \in \mathrm{On}} w_{i,\beta}$ and $r_{i,\alpha} = \lim_{\beta \in \mathrm{On}} r_{i,\beta}$ hold. □

Lemma 2.17 (Reflection lemma)**.** *If $\{e\}^X(w) = 0$ then there exists unboundedly many β with $\{e\}_\beta^{X_\beta}(w) = 0$.*

Proof. As the steps of algorithm 3 can be carried out by an OTM, there is an OTM A so that

$$\{A\}(\langle i, \alpha \rangle) = \begin{cases} 1 & \text{if } i < \omega \text{ and } R_i \text{ receives attention or is injured in stage } \alpha \\ 0 & \text{otherwise.} \end{cases}$$

Consider the first-order formula $\phi(i, \alpha)$ given by $\{A\}(\langle i, \alpha \rangle) = 0$ or, more formally

$$\text{“}i, \alpha \in \mathrm{On}\text{ ”} \wedge (\exists C)[\text{“}C \text{ is a computation for } A \text{ on input } \langle i, \alpha \rangle \text{ with output 0”}]$$

where the latter part can be expressed in the language of ZFC (with parameter A) by following definition 1.5, and the former is well-known to be definable in ZFC. By the settling lemma,

$$(\forall i < \omega)(\forall \alpha \in \mathrm{On})(\exists \beta)\, \phi(i, \alpha) = 1 \wedge (\forall \delta > \beta)\, \phi(i, \alpha, x)$$

holds, and so, by the replacement scheme, the collection of *settling points* $S = \{\beta :$ there is a requirement that first settles in stage $\beta\}$ is a set. In particular, it is bounded by $\alpha = \bigcup S + 1$. After stage α, no requirement receives attention, hence $A_\alpha = A$ and $B_\alpha = B$. So if $\{e\}^X(w) = 0$ then $\{e\}^X(w) = \{e\}_\beta^{X_\beta}$ for all $\beta > \alpha$. □

The preservation lemma is proved in the exact same way as for Turing machines (lemma 2.5).

Lemma 2.18 (Preservation lemma)**.** *Assume $R_i = R_e^X$ settles. If it is injured last, then $w_i \notin A \cup B$. If it receives attention last, then $\{e\}^X(w_i) = 0$ and $w_i \in Y$.*

The proof of the Friedberg-Muchnik theorem for OTMs then follows from the attention, reflection, preservation, and settling lemmas as on page 19.

Theorem 2.19 (Friedberg-Muchnik for OTMs, Hamkins and Miller)**.** *There are OTM-incomparable OTM-semi-decidable sets A,B.*

2.4 For α-machines

We shall sketch a proof for α-machines with $\omega < \alpha < \mathsf{On}$. Algorithm 3 can be reused, with one modification: if R_i receives attention in stage β, then we additionally require $r_{i,\beta} > \beta$.

Lemma 2.20. *For $\beta < \alpha$, stage β of the modified algorithm 3 is α-computable.*

Proof. Let e be an α-machine (or OTM) implied by algorithm 3 with the modification. It has to be shown that it takes it less than α steps to compute stage $\beta < \alpha$.

As α is admissible and thus L_α is closed under Σ_1-replacement with parameters, it suffices to show that going from one stage to the next takes less than α steps: there is a Σ_1-formula stating "there is a halting computation for $\{e\}(\delta)$", and so the result follows by induction: if there is a halting computation for all $\{e\}(\delta), \delta < \beta$, then there is a set of these halting computations in L_α, and the supremum of their lengths (again obtained by Σ_1-replacement and union) is an element of L_α and so lies below α. Then $\{e\}(\beta)$ halts since going from one stage to the next takes less than α steps and α is closed under addition.

To show that one can go from one stage to the next, by α's closure under ordinal arithmetic, it suffices to show that the individual lines of algorithm 3 are. In particular, as $\alpha > \omega$, the main loop (lines 5-15) is α-computable if the individual steps are. This follows from $\alpha \otimes \alpha \subseteq \alpha$, which holds since for all $\delta < \alpha$, it is easily seen that $\langle \delta, \delta \rangle$ is definable in $L_\delta \subset L_{\delta+1} \subset L_\alpha$. □

The attention and preservation, and finite injury lemmas are proved exactly as for OTMs. The settling lemma follows from the finite injury lemma the fact that α is closed under ordinal arithmetic. For the reflection lemma, however, extra care needs to be taken, as the supremum of the settling points (as defined in lemma 2.17) does not always lie below α.[6]We also cannot rely on the α-clockable ordinals as with ITTMs, since whether the α-clockable ordinals are bounded or unbounded in α depends on α.[7]

Lemma 2.21 (Reflection for α-machines). *If $\{e\}^A(w) = 0$, then there are cofinally many $\delta < \alpha$ with $\{e\}_\delta^{A_\delta}(w) = 0$. Similar for B.*

[6]This does sometimes hold, for instance if α is greater than the supremum of the settling points for OTMs. (Such admissible α exist, as regular cardinals are admissible.)

On the other hand, consider $\alpha = \Gamma$, which is shown to be admissible in [6, corollary 8.2]. If the supremum of the settling points would lie below α, then it would be ITTM-writable, and so A and B would be ITTM-decidable. But that would, by appendix B.2, imply that A and B are Γ-decidable, contradicting that they are incomparable.

[7]There are countably many OTM-clockable ordinals, so they have a supremum. As all halting α-machine computations are halting OTM-computations, any α above the OTM-clockable ordinals satisfies that the α-clockable ordinals are bounded in α.

On the other hand, by appendix B.2 the Γ-clockable ordinals are the ITTM-clockable ordinals, and so the Γ-clockable ordinals are cofinal in Γ.

Proof. Assume $\{e\}^A(w) = 0$. Let $R = \limsup_{\beta<\alpha} \max_{i<\omega} r_{i,\beta}$.

If $R < \alpha$, then—by the modification to the algorithm that ensures restraints made at stage β must be greater than β—there is a stage $\beta < \alpha$ after which no requirements receive attention, and so no requirement is injured either (since requirements are only injured if a requirement of higher priority receives attention). Hence $A_\beta = A$, and so $\{e\}_\delta^{A_\delta}(w) = \{e\}^A(w)$ for all $\delta > \beta$.

If not $R < \alpha$, then $R = \alpha$. As the computation $\{e\}^A(w) = 0$ halts, it takes strictly less than α steps, and so its queries are bounded by some $\beta < \alpha$. By $R = \alpha$, there is some stage $\varepsilon < \alpha$ so that $r_{i,\varepsilon} > \beta$ for some $i < \omega$. Then there is a stage $\zeta > \varepsilon$ in which all requirements R_j, $j < i$ that have not yet settled in stage ε have been injured or have received attention (recall, α is closed under addition), and so after step ζ no elements below β are added to $A \cup B$. Hence $\{e\}_\delta^{A_\delta}(w) = \{e\}^A(w) = 0$ holds for all $\delta > \zeta$. □

We thus have the attention, settling, preservation, and reflection lemmas needed to conclude a Friedberg-Muchnik theorem as on page 19.

Theorem 2.22. *For all admissible α, there are α-semi-decidable sets that do not α-reduce to each other.*

2.5 For p-OTMs

We sketch a proof for p-OTMs. While before it was convenient if the settling points could be shown to have a supremum, with parametrised machines, the settling points having a supremum implies failure, since then the constructed sets are actually decidable in the supremum (provided as a parameter). In a way, this brings us back closer to the original proof for Turing machines.

There are now class-many requirements

$$\{\varepsilon\}^A \neq B \tag{R_ε^A}$$

and

$$\{\varepsilon\}^B \neq A \tag{R_ε^B}$$

for all pairs $\varepsilon = \langle e, \pi\rangle$ of OTM and parameter, ordered $\{R_\varepsilon : \varepsilon \in \mathrm{On}\}$. This can be managed by modifying algorithm 3 so that at stage α only requirements R_ε with $\varepsilon < \alpha$ are considered, similar to algorithm 1. This way, the approximations $A_\varepsilon, B_\varepsilon$ are sets for all ε, and because all halting computations are also sets (implying the class of queries made in such a computation is a set as well), there are no further challenges in proving a reflection lemma.

Instead of a finite injury lemma, requirements can be shown to be injured boundedly often (the class of stages in which R_ε is injured forms a set), which is sufficient to show that for every requirement R_ε there is a point in time after which it cannot be injured. It follows that one can prove

Theorem 2.23. *There are parametrised OTM-semi-decidable classes incomparable by OTMs with parameters.*

However, a new issue arises in the approximation algorithm design. So far, no thought had to be spent on what happens with the witness attempts and restraints in limit stages: since they only change value finitely often, there is no limit behavior to speak of. Now there is.

A solution is found in [12], where a Friedberg-Muchnik theorem is proved for p-α-machines. First, to preserve that every ordinal is only ever a witness to at most one requirement, we again let requirement R_ε only take witness attempts of the form $\langle \varepsilon, \zeta \rangle$. If $w_{i,\alpha} = \langle \varepsilon, \zeta \rangle$, we write $w'_{i,\alpha}$ for ζ.

The variables are stored as follows: by use of a pairing function, a storage tape is divided into $\mathsf{On} \times 2$ tapes of ordinal length. At stage α the value of $r_{\varepsilon,\alpha}$ is stored on the $(\varepsilon, 0)$th tape, and $w'_{\varepsilon,\alpha}$ on the $(\varepsilon, 1)$th tape, so that a value of β is stored as β 1s followed only by 0s.

For each requirement, witness attempts only grow over time, hence the limit behavior dictated by the definition of p-OTMs makes it so that at limit stages α, $w'_{\varepsilon,\alpha}$ is the supremum of the $w'_{\varepsilon,\eta}$ that came before. Restraints, on the other hand drop to 0 on injury.

Ergo: if R_ε was injured cofinally often before time α, then $r_{\varepsilon,\alpha} = 0$ and $w'_{\varepsilon,\alpha}$ is the supremum of the $w'_{\varepsilon,\eta}$ before it, so $w_{\varepsilon,\alpha} = \langle \varepsilon, w'_{\varepsilon,\alpha} \rangle$ can be seen as a new witness attempt. Furthermore, since $w_{\varepsilon,\alpha}$ can only be a witness for R_ε, we know that it has not yet been added to $A_\alpha \cup B_\alpha$ and the proof proceeds as usual. If, on the other hand, R_ε was not injured cofinally often before time α, then at the start of stage α, the value of $w'_{\varepsilon,\alpha}$ is the limit of the $w'_{\varepsilon,\eta}$ before it, and likewise $r_{\varepsilon,\alpha}$ the limit of the $r_{\varepsilon,\eta}$ before it.

So the storage strategy from [12, section 4]—in particular, storing $w'_{\varepsilon,\alpha}$ instead of $w_{\varepsilon,\alpha}$—dictates the correct behavior on limit stages. See algorithm 4 for an overview of the approximation procedure.

With these changes, the proof of theorem 2.23 follows as sketched. Specifically, one can prove the following by induction.

Lemma 2.24 (Bounded injury). *For each requirement R_ε there is a stage $\alpha \in \mathsf{On}$ after which R_ε is never injured or receives attention.*

The reflection lemma would become:

Lemma 2.25 (Reflection lemma). *If $\{e\}^A(x) = 0$ then there is a stage $\delta \in \mathsf{On}$ after which $\{e\}_\zeta^{A_\zeta}(x) = 0$ for all $\zeta \geq \delta$.*

This can be proved by considering that all halting oracle computations have ordinal length, and hence the queries made in it are bounded. Analogues of the attention and preservation lemmas can also be obtained by considering the limit behavior of the storage strategy. In particular, the new case to consider is if α is the first stage so that $R_\varepsilon = (R_\varepsilon^A)$ is never injured in stages $\beta \geq \alpha$ and

Algorithm 4: Step $\alpha > 0$ of the Friedberg-Muchnik algorithm for p-OTMs.

```
$A_\alpha \leftarrow A_{<\alpha}$;    $B_\alpha \leftarrow B_{<\alpha}$
**for** $i < \alpha$ **do**
    $w'_{i,\alpha} \leftarrow \sup_{t<\alpha} w_{i,t}$;    $r_{i,\alpha} \leftarrow \liminf_{t<\alpha} r_{i,t}$
**end for**
**for** $i < \alpha$ **do**
    **let** $\varepsilon, X, Y$ such that $R_i = (R^X_\varepsilon)$ and $X \neq Y$
    $w_{i,\alpha} \leftarrow \langle i, w'_{i,\alpha} \rangle$
    **if** $\{\varepsilon\}^{X_\alpha}_\alpha(w_{i,\alpha}) = 0 \land w_{i,\alpha} \notin Y_\alpha$ **then**                ▷ $R_i$ *receives attention*
        $Y_\alpha \leftarrow Y_\alpha \cup \{w_{i,\alpha}\}$
        $r_{i,\alpha} \leftarrow \sup\{x : x$ is queried in $\{\varepsilon\}^{X_\alpha}_\alpha(w_{i,\alpha}) = 0\}$
        **for** $j \in (i, \alpha)$ **do**                                                ▷ $R_j$ is *injured*
            $w'_{j,\alpha} \leftarrow \min\{w' : (\forall k < j)\, \langle i, w' \rangle > \langle k, w'_{k,\alpha} \rangle, r_{k,\alpha}\}$
            $r_{j,\alpha} \leftarrow 0$
        **end for**
    **end if**
**end for**
$w'_{\alpha,\alpha} \leftarrow 0$
```

α is a limit. Then the value of $w'_{\varepsilon,\alpha}$ lies strictly above all $w'_{\varepsilon,\eta}, \eta < \alpha$, and so $w_{\varepsilon,\alpha} \notin B_{<\alpha}$. Hence it can still be proved that, if w is the final witness of R^A_ε, then $w \in B \leftrightarrow \{\varepsilon\}^A(w) = 0$. Theorem 2.23 follows.

In [12, section 4], Koepke and Seyfferth prove a Friedberg-Muchnik theorem for p-α-machines.

Chapter 3

The splitting theorem

Roughly, the splitting theorem states that every semi-decidable degree, except for the degree 0 of decidable sets, can be split into two strictly lower, incomparable semi-decidable degrees. An important corollary is that, because the splitting can be iterated, an infinite complete binary tree can be embedded in the partial order of semi-decidable degrees.

First, we discuss the formal statement of the theorem and this corollary. Then we prove the splitting theorem for TMs, followed by a proof of a splitting theorem for ITTMs restricted to ITTM-semi-decidable subsets of ITTM-decidable sets of writable reals, and show that this is sufficient for generalizing the corollary. The chapter concludes with a discussion of a failed attempt at generalizing the splitting theorem to OTMs, and a brief sketch for a splitting theorem for p-OTMs.

3.1 A tree of semi-decidable degrees

The following formulation of the splitting theorem for Turing machines is due to Shoenfield [19, section 14], who attributes the theorem to Sacks' [17].

Theorem 3.1 (The splitting theorem, Sacks, Shoenfield)**.** *If C is semi-decidable and D is simple, then C is the union of disjoint, semi-decidable A, B which D does not reduce to.*

Simple sets were defined by Post [15, section 5]. Dekker showed that every undecidable semi-decidable set is equivalent with a simple set [2, theorem 1].

Definition 3.2 (Post)**.** A set is *simple* if it is coinfinite, semi-decidable, and its complement does not contain any infinite semi-decidable set.

That C is the disjoint union of A and B is relevant because of the following.

Lemma 3.3. *If A and B are disjoint and M-semi-decidable, both reduce to $A \cup B$. (In fact, for those M for which $\leqslant_M$ is lifted to the M-degrees, the M-degree of $A \cup B$ is the least upper bound of the M-degrees of A and B.)*

Proof. To reduce A to $A \cup B$, an M-machine can act as follows: given x, check whether x is in $A \cup B$. If not, then x is not in A. Otherwise, x lies in either $A = \operatorname{dom}\{a\}$ or $B = \operatorname{dom}\{b\}$. To decide in which, the TM then simultaneously simulates $\{a\}(x)$ and $\{b\}(x)$ until one of the two computations halts.

(The second part follows from the observation that if $A, B \leqslant_M C$, then $A, B \leqslant_M \{\langle x, i\rangle : x \in A \wedge i = 0 \vee x \in B \wedge i = 1\} \leqslant_M C$.) $\square$

Hence, in the case that $C = D$ is a simple set, theorem 3.1 says that there are $A, B \leqslant_{\mathrm{TM}} C$ with $C \nleqslant_{\mathrm{TM}} A, B$. It follows that A, B do not reduce to each other: were $A \leqslant_{\mathrm{TM}} B$ (resp. $B \leqslant_{\mathrm{TM}} A$) to hold, then $C = A \cup B$ would readily reduce to B (resp. A). So the degree of C is nicely "split" in two incomparable lower degrees with least upper bound C. Combined with Dekker's theorem, every nonzero semi-decidable degree can be split.

3.2 For Turing machines

The proof in this section is based on that in [19, section 14]. It is a finite injury priority argument similar to the Friedberg-Muchnik one, with a twist. The requirements are

$$\{e\}^A \neq D \qquad (R_e^A)$$

$$\{e\}^B \neq D \qquad (R_e^B)$$

$$e \in C \leftrightarrow e \in A \cup B \qquad (R_e^C)$$

for all $e < \omega$. We fix a computable ordering $\{R_e : e < \omega\}$ of the R_e^A, R_e^B.

With the Friedberg-Muchnik theorem, the strategy was to search for computations of the form $\{e\}^A(w) = 0$ and then adding w to B, so that $\{e\}^A \neq B$. This time however, we want $\{e\}^A \neq D$, and D is given—we cannot add elements to it. It is not decidable either, meaning the contents of D are never fully known in any stage in a Turing machine computation.

However, D is semi-decidable. So a Turing machine could build increasing approximations of D while searching for computations of the form $\{e\}^A(w) = 0$. The idea is to preserve many such computations $\{e\}^A(w_n) = 0$ in the hope of eventually finding $w_n \in D$ for at least one n.

Because D is simple, this strategy cannot fail: if an algorithm tries infinitely many w_n, and all w_n would lie outside D, then the set of attempts would be an infinite semi-decidable subset of $\overline{D}$, which cannot be.

The construction stages of A, B are described in algorithm 5. The algorithm uses increasing approximations C_s, D_s of C and D. E.g. if $D = \operatorname{dom}\{d\}$, define $D_s = \{x < s : \{d\}_s(x)\downarrow\}$. Instead of keeping track of a single witness candidate per requirement, for each requirement R_i and stage s there is a set $W_{i,s}$ of witness candidates in stage s. Requirement (R_e^A) is always on the lookout for more candidates with $\{e\}_s^{A_s}(w) = 0$, unless one of its candidates is found to be in D_s.

To satisfy the requirements (R_e^C), elements in $C_s \setminus (A_s \cup B_s)$ are added to either A_s or B_s. Adding e to A_s may injure possible witness computations $\{m\}^{A_s}(w) = 0$ of requirements (R_m^A) whose restraint exceeds e, and similarly adding e to B_s may injure requirements (R_m^B). Hence, often either some (R_m^A) or an $(R_{m'}^B)$ must be injured. Then the requirement of highest priority is spared.

Algorithm 5: Step s of the splitting algorithm.

```
$A_s \leftarrow A_{<s}$;    $B_s \leftarrow B_{<s}$
for $i < s$ do
    $W_{i,s} \leftarrow \lim_{t<s} W_{i,t}$;    $r_{i,s} \leftarrow \lim_{t<s} r_{i,t}$
end for
for $i < s$ do
    if $W_{i,s} \cap D_s = \varnothing$ then
        let $e, X, Y$ such that $R_i = (R_e^X)$ and $X \neq Y$
        if there is a least $w \in s \setminus W_{i,s}$ with $\{e\}_s^{X_s}(w) = 0$ then
                                                        ▷ $R_i$ receives attention
            $W_{i,s} \leftarrow W_{i,s} \cup \{w\}$
            $r_{i,s} \leftarrow \sup\{r_{i,s}, w+1, \sup\{x+1 : x$ is queried in $\{e\}_s^{X_s}(w) = 0\}\}$
        end if
    end if
end for
for $i < s$ do
    if $i \in C_s \setminus (A_s \cup B_s)$ then                ▷ $(R_i^C)$ receives attention
        if $j = \min\{k : r_k > i\}$ exists then
            let $e, X, Y$ such that $R_j = (R_e^X)$ and $X \neq Y$
            $Y_s \leftarrow Y_s \cup \{i\}$
            for $k \in (j, s)$ do                               ▷ $R_k$ is injured
                $W_{k,s} \leftarrow \varnothing$;    $r_{k,s} \leftarrow 0$
            end for
        else
            $A_s \leftarrow A_s \cup \{i\}$
        end if
    end if
end for
```

3.2.1 Correctness

Given C, D, it is clear that A, B determined by algorithm 5 satisfy the requirements (R_e^C). A finite injury argument is used to prove satisfaction of the R_i.

Lemma 3.4 (Finite injury). *All R_i* settle*: there is a (least) stage t after which R_i is never injured nor receives attention; $r_i = \lim_{s<\omega} r_{i,s}$ and the pointwise limit $W_i = \lim_{s<\omega} W_{i,s}$ exist and are equal to $r_{i,t}, W_{i,t}$ respectively.*

Proof. Note that requirements can only be injured by requirements of higher priority, of which there are always finitely many. If all requirements R_j with $j < i$ settle, then there is a stage s after which R_i is never injured. (Namely, once all elements of C below the $r_j, j < i$ have been added to A.) Then $\{W_{i,t} : t > s\}$ is an increasing sequence whose union W_i is either finite or infinite. It cannot be infinite, since if it were, it would by design of algorithm 5 (no elements are added to $W_{i,t}$ if $W_{i,t} \cap D_t \neq \varnothing$) be an infinite semi-decidable subset of $\overline{D}$, contradicting that D is simple. Hence W_i is finite, and R_i has settled in the first stage $t > s$ with $W_{i,t} = W_i$. By induction, all requirements settle. □

Lemma 3.5 (Reflection). $\{e\}^A(w) = 0 \rightarrow (\exists s)(\forall t > s)\, \{e\}_t^{A_t}(w) = 0$*, and similarly for B.*

Proof. Consider that the part of A (or B) used in the computation is finite: if u lies above the maximum value queried in $\{e\}^A(w) = 0$, then the first s with $A_s \cap u = A \cap u$ is as desired. (s exists because for each $x \in A \cap u$ there is a stage in which it is added to A, and there are finitely many such x.) □

Together, finite injury and reflection prove that all requirements are satisfied.

Proof of theorem 3.1. Let A, B be defined as by algorithm 5. They are semi-decidable, and $C = A \cup B$. Assume that that $R_i = (R_e^A)$ is not satisfied. Then $\{e\}^A = D$. Let s be a state at which R_i and all requirements of higher priority have settled. Then (by the finite injury lemma) we have $W_{i,s} = W_{i,t}$ for all $t \geq s$. By design of the algorithm, if $x \in W_{i,t}, t \geq s$ then $\{e\}^A(x) = \{e\}_t^{A_t}(x) = 0$, and so $W_{i,t} \cap D = \varnothing$ for all $t \geq s$. Because $\overline{D}$ is infinite and W_i is finite, there is a least $w \in \overline{D} \setminus W_i$, for which $\{e\}^A(w) = 0$ holds by assumption. Using the reflection lemma, there is a stage $t > s, w$ with $\{e\}_t^{A_t}(w) = 0$. The design of algorithm 5 now dictates that eventually, R_i receives attention and a new is element is added to $W_{i,t}$, contradicting that R_i had settled at stage s. □

While the proof above is also a finite injury argument, one way it may be seen as more complex than the Friedberg-Muchnik proof is that there is no computable function $I(i)$ bounding the number of injuries sustained by R_i [20, p. 65]—since it depends on the undecidable C, D—while such a function does exist for the Friedberg-Muchnik argument [14, exercise X.2.6].

As a generalization of theorem 3.1 would be meaningless without a matching generalization of Dekker's theorem, we discuss its proof as well.

Theorem 3.6 (Dekker). *Every undecidable semi-decidable set is equivalent with a simple set. (All nontrivial semi-decidable degrees contain simple sets.)*

Proof, as presented in [19, section 12]. Assume A is undecidable and semi-decidable. We must find a TM-equivalent simple set S. Define

$$S = \{n : (\exists m > n)\, a_m < a_n\}$$

where $\{a_n : n \in \omega\}$ is a computable enumeration of A. S is semi-decidable by a TM that, given n, tries all $m < \omega$ until it finds one with $a_m < a_n$. S reduces to A, since $n \in S$ if and only if there is a $k < a_n$ in $A \setminus \{a_m : m < n\}$. Given x, the $n > x$ with the smallest a_n always lies outside S, so S is coinfinite.

Assuming $n \in \overline{S}$ and $m < a_n$, we have $m \in A \leftrightarrow m \in \{a_k : k < n\}$. Since S is coinfinite, such an n can be found for every m by enumerating $\overline{S}$, and so A reduces to S.

Likewise: if $\overline{S}$ were to contain an infinite semi-decidable set X, then one could find such an n for every m by enumerating X, which would make A decidable, contradicting the assumption on A. □

Corollary 3.7. *Every nonzero semi-decidable degree splits.*

Corollary 3.8. *There is a complete infinite binary tree of semi-decidable degrees.*

3.3 For ITTMs

A direct translation of the thickness lemma to ITTMs would be: if $C \subseteq 2^\omega$ is ITTM-semi-decidable and D is "simple", then there are disjoint ITTM-semi-decidable A, B with $A \cup B = C$ so that D does not ITTM-reduce to either—where "simple" would mean that D is ITTM-semi-decidable and intersects all infinite ITTM-semi-decidable sets. Whether this holds, we do not know.

To prove the above with a priority argument similar to that in section 3.2, there would at least have to be a ITTM-computably well-ordered set $S \subseteq 2^\omega$ of stages, and ITTM-computable approximations C_s with $C = \bigcup_{s \in S} C_s$, and likewise for D. Furthermore, S would have to be generable for A and B to be semi-decidable. This restricts S to the accidentally writable reals. But then the contents of C_s, ITTM-generable in the argument $s \in S$, must be accidentally writable as well.

So, using a priority argument as in section 3.2, one could only prove a splitting theorem for C, D in the accidentally writable reals. Furthermore, the argument in section 3.2 uses that bounded unions $\cup_{t<s} W_{i,t}$ of witness attempt sets are decidable, and unbounded unions $\cup_{t>s} W_{i,t}$ semi-decidable. If t is not writable, this is not apparent. All in all, if a general splitting theorem were to hold, its proof would not be similar to the classical priority argument proof in section 3.2.

In this section, we shall prove a splitting theorem for certain sets $C, D \subseteq \mathsf{Won}$ and a matching generalization of Dekker's theorem, which together prove one of the main consequences of a general splitting theorem: that a complete infinite binary tree can be embedded in the semi-decidable degrees.

In this context, a straightforward translation from "simple" for would be

Definition 3.9. $A \subseteq \mathsf{Won}$ is Won-*simple* if it is ITTM-semi-decidable, cocofinal in Won, and $\mathsf{Won} \setminus A$ does not contain cofinal ITTM-semi-decidable sets.

However, we do not know whether Won-simple sets exist. As the clockable ordinals are ITTM-decidable (in contrast to the set of writable codes for writable ordinals, cf. appendix A.2), we are able to prove existence for the following, and this translation of “simple” suffices for our theorem below.

Definition 3.10. $A \subseteq \mathsf{Clk}$ is Clk-*simple* if it is semi-decidable, cocofinal in Clk, and $\overline{A} \cap \mathsf{Clk}$ does not contain unbounded semi-decidable sets.

We further adapt two notions from α-recursion theory, used in [20] for a splitting theorem. A set $A \subseteq \alpha$ is α-finite if it is an element of L_α [18, p. 155], and regular if $A \cap \delta$ is α-finite for all $\delta < \alpha$ [18, section VII.3.5]. In [12, theorem 12] it is shown that $A \subseteq \alpha$ is α-finite if and only if A is bounded and p-α-decidable. This leads us to the following definition.

Definition 3.11. $A \subseteq \mathsf{Won}$ is *ITTM-finite* if it is bounded in Won and ITTM-decidable. $A \subseteq \mathsf{Won}$ is *(ITTM-)regular* if it is ITTM-semi-decidable, and all intersections with ITTM-finite sets are ITTM-finite.

From appendix B.2 it can be concluded that the ITTM-finite sets are, up to coding, precisely the Γ-finite sets, but this fact shall not be used. In this section, we shall often write “regular” for ITTM-regular and “simple” for Clk-simple.

Lemma 3.12. *$A \subseteq \mathsf{Won}$ is regular if and only if it is semi-decidable and all intersections $A \cap \alpha, \alpha \in \mathsf{Won}$ are decidable.*

Proof. Every $\alpha \in \mathsf{Won}$ is decidable: given x, reject if x does not code an ordinal. If it codes an ordinal β, reject if $\beta \geq \alpha$. Otherwise, β is a writable ordinal. Compute the canonical code for β, and accept x if and only if x is that canonical code. Furthermore, all $\alpha \in \mathsf{Won}$ are bounded by definition, and so they are ITTM-finite. It follows that if A is regular, all intersections $A \cap \alpha, \alpha \in \mathsf{Won}$ are ITTM-finite, by definition of regularity.

Now assume that $A \subseteq \mathsf{Won}$ is semi-decidable, and that all intersections $A \cap \alpha, \alpha \in \mathsf{Won}$ are ITTM-finite. Let B be ITTM-finite. Then it is bounded, so there is a $\beta \in \mathsf{Won}$ strictly above all elements of B. By assumption, $A \cap \beta \supseteq B$ is ITTM-finite. Because B is decidable, there is a machine e with $\{e\} = B$, and because $A \cap \beta$ is decidable, there is a machine m with $\{m\} = A \cap \beta$. Then the machine that, on input x, outputs 1 if $\{e\}(x) = \{m\}(x) = 1$ and 0 otherwise, decides B. So $A \cap B$ is ITTM-finite, which concludes the proof. □

Whether regular and simple sets are as ubiquitous as TM-simple sets are in ordinary computability, we do not know, but we will show that they appear relatively often at the end of this section. First, we prove

Theorem 3.13. *Let C be regular, semi-decidable, and D simple. Then there are disjoint regular semi-decidable A, B so that $C = A \cup B$ and D does not reduce to either.*

The regularity assumption on C is taken from [20], though the proof below is rather based on that for Turing machines. We use Clk-simplicity here, but the same proof would work for ITTM-simple sets D, replacing Clk with Won.

3.3.1 The algorithm

The incomparable A, B are defined by a procedure akin to algorithm 5, with modifications so that the storage of witness attempts is consistent across limit steps. This is done as follows: there is a witness storage tape with the property that (during stage α) $w \in W_{i,\alpha}$ if and only if the $\langle i, e\rangle$th cell of the tape contains a 1, where e is machine that writes w (the first that does it in the least amount of time). Restraints $r_{i,\alpha}$ are stored similarly on a restraint tape, where at stage α the $\langle i, e\rangle$th cell contains a 1 if and only if $\{e\}(0)$ writes an ordinal $\beta < r_{i,\alpha}$ (and e is the least machine to do so in the least amount of time.)[8] Furthermore, the algorithm is modified so that, between injuries, elements added to a witness attempt set are strictly increasing over time.

With these technical changes, at the start of stage α, the storage tape always contains the pointwise limits of the $W_{i,\beta}$ that came before, and the limit inferiors of the $r_{i,\alpha}$ that came before. As restraints increase between injuries, at the start successor stages $\alpha + 1$ we have $r_{i,\alpha+1} = r_{i,\alpha}$, and at start of limit stages λ we have $r_{i,\lambda} = 0$ if R_i was injured cofinally often before stage λ, or $r_{i,\lambda} = \lim_{\alpha<\lambda} r_{i,\alpha}$ otherwise.

See algorithm 6 for an overview of the changes. In it, s-step approximations of C_s, D_s as defined in definition 1.40 are used. Recall that by proposition 1.42, since $C, D \subseteq \mathsf{Won}$, we have $\beta \in C \rightarrow (\exists s \in \mathsf{Won})\, \beta \in C_s$ and similarly for D.

From the storage strategy described above and the properties in section 1.4, it follows that algorithm 6 can be carried out by an ITTM. In particular, lines 1–4 are consequences of the storage strategy, and bounded loops and simulations are ITTM-computable by section 1.4. Only line 11, that is

$$r_{i,s} \leftarrow \sup \{r_{i,s}, w + 1, \sup \{\alpha + 1 : \alpha \in \mathsf{Clk} \text{ is queried in } \{e\}_s^{X_s}(w) = 0\}\}$$

requires scrutiny. The queries made are decidable by simulation, and suprema can be found by generating Won in-order and searching for the first upper bound. It has to be shown, however, that $\{\alpha + 1 : \alpha \in \mathsf{Clk} \text{ is queried in } \{e\}_s^{X_s}(w) = 0\}$ is bounded. By contradiction: if it ever were unbounded, let $s \in \mathsf{Won}$ and $e, i < \omega$ be the first for which it occurs. Because s, e, i are writable, there is an ITTM which follows algorithm 6 up to having to compute line 11 for this specific case. It then simulates $\{e\}_s^{X_s}(w) = 0$, clocks all ordinal queries made, and halts, halting after Γ or more steps. (Yet its halting time must be a clockable ordinal.)

3.3.2 Correctness

Lemma 3.14 (Bounded injury lemma)**.** *All requirements settle in clockable (writable) time: for each R_i there is an $\alpha \in \mathsf{Won}$ such that R_i is never injured after stage α, and we have $r_i = \lim_{\beta\in\mathsf{Won}} r_{i,\beta} = r_{i,\alpha}$ and $W_i = \bigcup_{\alpha<\beta\in\mathsf{Won}} W_{i,\beta} = W_{i,\alpha}$. In particular, W_i is ITTM-finite.*

[8]Alternatively, any storage technique could be used for the restraints, and then at limit steps the limit inferiors could be computed from the witness sets.

Algorithm 6: Stage $s < \Gamma$ of the splitting algorithm for ITTMs.

```
1:  $A_s \leftarrow A_{<s}$;    $B_s \leftarrow B_{<s}$
2:  **for** $i < s$ **do**
3:      $W_{i,s} \leftarrow \lim_{t<s} W_{i,t}$;    $r_{i,s} \leftarrow \liminf_{t<s} r_{i,t}$
4:  **end for**
5:  **for** $i < \omega$ **do**
6:      **if** $W_{i,s} \cap D_s = \varnothing$ **then**
7:          **let** $e, X, Y$ such that $R_i = (R_e^X)$ and $X \neq Y$
8:          **if** there is a least $w \in \mathsf{Clk} \cap s \setminus (\sup W_{i,s} + 1)$ with $\{e\}_s^{X_s}(w) = 0$ **then**
9:                                                              $\triangleright$ $R_i$ *receives attention*
10:             $W_{i,s} \leftarrow W_{i,s} \cup \{w\}$
11:             $r_{i,s} \leftarrow \sup\{r_{i,s}, w+1, \sup\{\alpha+1 : \alpha \in \mathsf{Clk}$ is queried in $\{e\}_s^{X_s}(w) = 0\}\}$
12:         **end if**
13:     **end if**
14: **end for**
15: **for** $i < s$ **do**
16:     **if** $i \in C_s \setminus (A_s \cup B_s)$ **then**                 $\triangleright$ $(R_i^C)$ receives attention
17:         **if** $j = \min\{k : r_k > i\}$ exists **then**
18:             **let** $e, X, Y$ such that $R_j = (R_e^X)$ and $X \neq Y$
19:             $Y_s \leftarrow Y_s \cup \{i\}$
20:             **for** $k \in (j, \omega)$ **do**                      $\triangleright$ $R_k$ is *injured*
21:                 $W_{k,s} \leftarrow \varnothing$;    $r_{k,s} \leftarrow 0$
22:             **end for**
23:         **else**
24:             $A_s \leftarrow A_s \cup \{i\}$
25:         **end if**
26:     **end if**
27: **end for**
```

Proof. By induction. Assume all requirements of higher priority than R_i settle. Then there is a stage $\delta' < \Gamma$ in which all R_j, $j < i$ have settled. (There are finitely many such R_j and Γ is closed under addition [6, corollary 8.2].)

Let $r = \max_{j<i} r_j$. Define $C' = C \cap r$. It is ITTM-finite by the regularity of C (lemma 3.12). Hence, because $C' \subseteq \mathsf{Won}$ is bounded and decidable, and Won is generable in-order, there is an ITTM which, on input 0, finds for each $c \in C'$ the least $s_c < \Gamma$ with $c' \in C_{s_c}$, clocks these s_c, and then halts. (The s_c exist by proposition 1.42.) Its halting time ε lies beyond the supremum of these moments of first appearance s_c, and at the same time necessarily below Γ (because its halting time is a clockable ordinal). So let $\delta \in \mathsf{Won}$ be greater than ε and δ'. At stage δ, all requirements R_j, $j < i$ have settled, and $C' \subseteq A_\delta \cup B_\delta$ (since in every stage α, all elements of C_α are added to $A_\alpha \cup B_\alpha$).

It follows that R_i is never injured after stage δ, as R_i is only injured in stage $\alpha > \delta$ if there exists an $\zeta < \alpha$ with $\zeta \in C_\alpha \setminus (A_\alpha \cup B_\alpha)$ and $r > \zeta$, but such i are elements of $C' \subseteq A_\delta \cup B_\delta \subseteq A_\alpha \cup B_\alpha$. Hence $\delta < \alpha < \beta$ implies $W_{i,\alpha} \subseteq W_{i,\beta}$.

Define $W_i = \bigcup_{\alpha>\delta} W_{i,\alpha}$. It remains to show that there is some $\upsilon < \Gamma$ with $W_{i,\upsilon} = W_i$: that then W_i is ITTM-finite follows from the observation $W_{i,\upsilon} \subseteq \upsilon$ (by line 8 of algorithm 6) and that there is an ITTM which decides W_i by constructing υ stages. (This ITTM exists because υ is writable, and algorithm 6 can be carried out by an ITTM.) That then $r_i = r_{i,\upsilon}$ follows from the observation that, at successor steps, restraints are only modified when changes are made to their witness sets.

If $W_i \cap D \neq \varnothing$, then (by definition of W_i and proposition 1.42) there is some $\upsilon < \Gamma$ with $\upsilon > \delta$ and $W_{i,\upsilon} \cap D_\upsilon \neq \varnothing$. By the design of the algorithm (line 6), and since R_i is never injured after stage δ, no element is added to the witness set of R_i after stage υ. Thus $\alpha > \upsilon$ implies $W_{i,\alpha} = W_{i,\upsilon}$ as desired.

If on the other hand $W_i \cap D = \varnothing$, then $W_i \subseteq \mathsf{Clk} \setminus D$, and so W_i must be bounded since it is semi-decidable and D is simple. (It is semi-decidable by the computability of the algorithm and writability of δ: there is an ITTM which, given x, continuously constructs stages and halts if and only if there is a stage $\alpha > \delta$ with $x \in W_{i,\alpha}$.) So there exists a least $\kappa < \Gamma$ so that $\alpha \in W_i \to \alpha < \kappa$. It follows that W_i is decidable: given x, reject if it does not code a clockable ordinal, and if it is not a canonical code for a clockable ordinal. If it is a canonical code for a clockable ordinal α, reject x if $\alpha \geq \kappa$ (an ITTM can check this since κ is writable). Otherwise, since elements are added to W_i in increasing order (line 8), there are two cases: either there is a stage $\beta > \delta$ with $\alpha \in W_{i,\beta}$, in which case $\alpha \in W_i$, or there is a stage $\beta > \delta$ and an $\alpha' > \alpha$ with $\alpha' \in W_{i,\beta} \wedge \alpha \notin W_{i,\beta}$, in which case $\alpha \notin W_i$. So finally $\alpha \in W_i$ is decided by generating increasing approximations of $W_{i,\beta}, \beta > \delta$ until one of those two cases occurs.

But then W_i is ITTM-finite and thus, by a moment-of-appearance argument as for C' above, there must be some $\upsilon < \Gamma$ with $W_{i,\upsilon} = W_i$. □

Indeed, this could be called an ITTM-finite injury argument.

Lemma 3.15 (Reflection lemma). *If $w \in \mathsf{Won}$ and $\{e\}^A(w) = 0$, then there are cofinally many $\alpha \in \mathsf{Won}$ with $\{e\}^{A_\alpha}_\alpha(w) = 0$. Similarly for B.*

Proof. See the proof of lemma 2.9. □

Lemma 3.16 (Preservation lemma). *Let δ be the least stage so that $R_i = R_e^A$ is never injured in stages $\alpha \geq \delta$. Then $\{e\}^A(w) = 0$ holds for all $w \in W_i = \bigcup_{\alpha \geq \delta} W_{i,\alpha}$. Similarly for B.*

Proof. If δ is a limit, then R_i was injured cofinally often before stage δ, and so at the start of stage δ we have $r_{i,\delta} = 0$ and $W_{i,\delta} = \varnothing$. If δ is a successor, then R_i was last injured in stage $\delta - 1$, and so (because in every stage, no witness attempts are added after a requirement is injured, see algorithm 6) $W_{i,\delta-1} = \varnothing$ and $r_{i,\delta-1} = 0$ and at the start of stage δ we have $r_{i,\delta} = 0$ and $W_{i,\delta} = \varnothing$.

For each $\alpha \geq \delta$, elements are added to $W_{i,\alpha}$ if and only if $\{e\}^{A_\alpha}(w) = 0$, and then $r_{i,\alpha}$ is pushed above all queries made in this computation. no element below these restraints is ever added to A, as this would injure R_i. Hence all computations for $w \in W_{i,\alpha}$ are preserved: $\{e\}^A(w) = \{e\}^{A_\alpha}(w) = 0$. □

The rest of the proof is similar to the proof of theorem 3.1, here using that the W_i are ITTM-finite instead of finite.

Proof of theorem 3.13. Let A, B be defined as by algorithm 5. They are semi-decidable (by the algorithm and the fact that Won is generable), and $C = A \cup B$. Assuming that $R_i = (R_e^A)$ is not satisfied, $\{e\}^A = D$, let σ be a state at which R_i and all requirements of higher priority have settled, which exists by the bounded injury lemma. Then $(\forall \alpha > \sigma)\ W_{i,\alpha} = W_{i,\sigma} = W_i$ holds.

If a $w \in W_i \cap D$ were to exists then, by the preservation lemma, $\{e\}^A(w) = 0$ would hold, yet by assumption $\{e\}^A(w) = D(w) = 1$. So $W_i \cap D$ is empty, hence $W_{i,\alpha} \cap D_\alpha = \varnothing$ holds for all $\alpha \geq \sigma$.

As D is simple and hence cocofinal in Clk, and W_i is bounded (it is ITTM-finite by the bounded injury lemma), there are $w \in \mathsf{Clk} \setminus (\sup W_i + 1)$ with $w \notin D$ and thus (by the assumption $\{e\}^A = D$) with $\{e\}^A(w) = 0$. By the reflection lemma, there are hence $\tau > \sigma$ and $w \in \mathsf{Clk} \cap \tau \setminus (\sup W_i + 1)$ with $\{e\}_\tau^{A_\tau}(w) = 0$. But then for the first such (τ, w) it would be the case that $w \notin W_i$ would be added to $W_{i,\tau}$ (line 8 of algorithm 6), contradicting that R_i settled in stage σ.

Hence R_i is satisfied, and, by an analogous argument, the requirements $R_j = (R_e^B)$ are also satisfied. In every stage α, all elements of C_α are added to $A_\alpha \cup B_\alpha$, so (by proposition 1.42), requirements (R_e^C) are satisfied as well. So A and B satisfy all requirements: they are disjoint, their union is C, and D does not reduce to either.

Finally, A and B are regular because C is regular: if $\alpha \in \mathsf{Won}$ then $A \cap \alpha$ can be decided by, given x, first rejecting if $x \notin C \cap \alpha$, which can be done since $C \cap \alpha$ is ITTM-finite, and otherwise generating both A and B simultaneously until x appears in one of them (which it must). □

Corollary 3.17. *If C is regular and simple, then there are regular disjoint A, B with $A \cup B = C$ that C does not reduce to.*

Corollary 3.18. *If a degree contains a regular simple set, it can be split in two. (And those degrees can again be split in two.)*

3.3.3 Regular and simple sets

Degrees that contain regular simple sets can be split. We now turn to the question of which degrees contain regular simple sets. We shall only give a partial answer, by showing that every semi-decidable $A \subseteq B \subseteq \mathsf{Wrr}$ contained in a decidable B is equivalent with a regular set, and that every such A that is undecidable is equivalent with a regular simple set. An examples of such an A is the weak halting problem $h = \{e : \{e\}(0)\downarrow\}$, as it is contained in the decidable Clk.

The assumption that A is a subset of a decidable set B is used to adapt results from α-recursion theory, where every nonzero degree contains a regular simple set. In α-recursion theory, all inputs are writable ordinals, while with ITTMs it is hard to design reduction machines that do not fail when presented with, for example, a code for an (accidentally writable) ordinal above Γ. The decidable B allows for the rejection of such ordinals.

We first focus on subsets $A \subseteq B \subseteq \mathsf{Won}$, and then generalize to $A \subseteq B \subseteq \mathsf{Wrr}$.

Lemma 3.19. *If $A \subseteq \mathsf{Won}$ is semi-decidable and cofinal in Won, there is a computable bijection $\mathsf{Won} \to A$.*

Proof. Let $\{e\}$ be a machine with $\operatorname{dom}\{e\} = A$. For each writable ordinal α, define $A_\alpha = \{\beta < \alpha : \{e\}_\alpha(\beta)\downarrow\}$ (in canonical codes). Each A_α is ITTM-finite, and in fact there is an ITTM which, given a code for a writable α, decides A_α. Note: $\alpha < \beta \to A_\alpha \subseteq A_\beta$.

Define a well-order $\prec$ on A so that $\alpha \prec \beta$ if and only if

- the first δ with $\alpha \in A_\delta$ lies strictly below the first ε with $\beta \in A_\varepsilon$, or
- $\delta = \varepsilon$ and $\alpha < \beta$.

Since ITTMs can generate Won, there is an ITTM which, given two writable α, β, decides $\alpha \prec \beta$.

Define the partial function f recursively by $f(\alpha) = \min_\prec A \setminus f[\alpha]$. From the discussion above, f is ITTM-computable: the minimum, if it exists, can be found by generating Won and searching for the first nonempty $A_\beta \setminus f[\alpha]$ for increasing writable ordinals β.

If α is writable then, since A_α is ITTM-finite and $\prec$ is decidable, the order type of $(A_\alpha, \prec)$ is a writable ordinal: there is an ITTM which (on input 0) takes $\prec$-minimal elements out of A_α until it is exhausted and then halts. It follows that the order type of $(A, \prec)$ is either Γ or lower. Hence the range of f is A, and its domain δ is at most Γ.

If $\delta < \Gamma$ were to hold, then there would be an ITTM which (on input 0) halts after computing and clocking all elements of $f[\delta] = A$, taking more than $\sup A = \Gamma$ steps. So $\delta = \Gamma$. (The domain of a machine computing f can be restricted to Won by a.o., given x, searching for a computation that writes x.) □

Lemma 3.20. *There is an order preserving bijection* $\mathsf{Clk} \to \mathsf{Won}$.

Proof. Define the partial f by $f(\alpha) = \min \mathsf{Won} \setminus f[\mathsf{Clk} \cap \alpha]$. Since Clk is decidable and Won is generable in-order, f is computable. Since f is an order-preserving injection and since Clk and Won have the same order type [6, theorem 3.8], the domain of f is Clk and the range is Won. □

Corollary 3.21. *If* $A \subseteq \mathsf{Won}$ *is semi-decidable and cofinal in* Won*, there is a computable bijection* $\mathsf{Clk} \to A$.

The following is adapted from α-recursion theory [18, theorem 4.4.2].

Lemma 3.22. *If* $A \subseteq \mathsf{Clk}$ *is semi-decidable and cofinal, then it is equivalent with a regular* $B \subseteq \mathsf{Clk} \otimes \mathsf{Clk}$.

Proof. Let $A \subseteq \mathsf{Clk}$ be semi-decidable and cofinal. Then there is a computable bijection $f : \mathsf{Clk} \to A$ Let $p : \mathsf{Clk} \to \omega$ be a computable injection. (Sending ordinals to machines that clock them, for example.) Define

$$B = \{\langle w, x\rangle : w, x \in \mathsf{Clk} \wedge (\exists y \in \mathsf{Clk})\, x < y \wedge p(f(y)) < p(f(x)) \wedge f(y) < w\}.$$

B is semi-decidable: given a real z, a machine searches for $x, w \in \mathsf{Won}$ so that z codes $\langle x, w\rangle$. If such x, w exist, they are eventually found, and the machine then searches for a $y \in \mathsf{Won}$ satisfying $x < y \wedge p(f(y)) < p(f(x)) \wedge f(y) < w$ and halts if one is found. Since p, f are computable, y is found if and only if it exists.

Define the approximations

$$B^t_y = \{\langle w, x\rangle < \langle t, t\rangle : w, x \in \mathsf{Clk} \wedge x < y \wedge p(f(y)) < p(f(x)) \wedge f(y) < w\}.$$

for all $t, y \in \mathsf{Clk}$. They are not monotonically increasing in y, but $B \cap \langle t, t\rangle$ is their union.

Each individual approximation is uniformly decidable in y and t: there is an ITTM that, given canonical codes for $t, y \in \mathsf{Clk}$, and a real z, decides $z \in B^t_y$ by iterating over all $w, x < t$, checking $w, x \in \mathsf{Clk} \wedge \langle w, x\rangle < t$, whether z is a canonical code for any $\langle w, x\rangle$ found and finally, seeing if w, x are as desired.

Therefore, if there were a $\beta \in \mathsf{Won}$ with $B \cap \langle t, t\rangle = \cup_{\alpha<\beta} B^t_\alpha$, then $B \cap \langle t, t\rangle$ would be ITTM-finite.

Hence, assuming towards a contradiction that $B \cap \langle t, t\rangle$ is not ITTM-finite, there is in particular an infinite sequence $\{y_i : i < \omega\}$ in $\mathsf{Won} \setminus t$ such that

$$i < j \to y_i < y_j \wedge B^t_{y_i} \not\supseteq B^t_{y_j}.$$

Note that if $t \leq y_i < y_j$ and $B^t_{y_i} \not\supseteq B^t_{y_j}$ then there is some $\langle w, x\rangle \in \langle t, t\rangle$ so that

$$[\langle w, x\rangle \in B^t_{y_j}] \wedge [\langle w, x\rangle \notin B^t_{y_i}]$$

equivalently

$$[p(f(y_j)) < p(f(x)) \land f(y_j) < w] \land \neg[p(f(y_i)) < p(f(x)) \land f(y_i) < w]$$

holds, which implies

$$[p(f(y_j)) < p(f(x)) \land p(f(y_i)) \not< p(f(x))] \lor [f(y_j) < w \land f(y_i) \not< w]$$

and so

$$p(f(y_j)) < p(f(y_i)) \lor f(y_j) < f(y_i)$$

holds. Hence $i < j$ implies $p(f(y_j)) < p(f(y_i)) \lor f(y_j) < f(y_i)$. Since $p[\mathsf{Clk}]$ is a subset of ω, the sequence $\{f(y_i) : i < \omega\}$ contains an infinite strictly decreasing subsequence of ordinals: contradiction.

So $B \cap \langle t, t\rangle$ is ITTM-finite for all $t \in \mathsf{Won}$: B is regular.

It remains to be proved that $A \equiv_{\mathrm{ITTM}} B$. Because Clk is decidable and $A \subseteq \mathsf{Clk}$, it suffices to show that A and $\mathsf{Clk} \setminus A$ are semi-decidable in B. Then A can be reduced to B by, given z, rejecting if z does not code a clockable ordinal, or if it does code a clockable ordinal but is not its canonical code. (Recall that, if x codes a writable ordinal, its canonical code is computable in x.) Finally, if z is a canonical code for a clockable ordinal α, either $\alpha \in A = \operatorname{dom}\{e\}^B$ or $\alpha \in \mathsf{Clk} \setminus A = \operatorname{dom}\{e'\}^B$, and this can be decided by simultaneously simulating both $\{e\}^B(z)$ and $\{e'\}^B(z)$: one of them must halt.

We first prove that $\mathsf{Clk} \setminus A$ is semi-decidable in B. Assuming $z \in A \setminus f[x]$ and $\langle z+1, x\rangle \notin B$ for some $x \in \mathsf{Clk}$, we find (by substituting $y = f^{-1}(z)$ in the definition of B) that $p(z) \geq p(f(x))$ holds. Hence

$$p(z) < p(f(x)) \land \langle z+1, x\rangle \notin B \to z \notin A \setminus f[x]$$

holds, and so, for all $z \in \mathsf{Clk}$ the right-to-left-direction of

$$z \notin A \leftrightarrow (\exists x \in \mathsf{Clk})\ p(z) < p(f(x)) \land \langle z+1, x\rangle \notin B \land z \notin f[x] \tag{3.1}$$

holds. The other direction: given z, there is a y_0 with $(\forall x > y_0)\ p(z) < p(f(x))$ since $p(z) < \omega$, and then the x with $p(f(x)) = \min\{p(f(x')) : x' > y_0\}$ satisfies $p(z) < p(f(x)) \land (\forall y > x)\ p(f(y)) \not< p(f(x))$, so $p(z) < p(f(x)) \land \langle z+1, x_0\rangle \notin B$.

From (3.1) and the fact that ITTMs can generate Clk, the semi-decidability of $\mathsf{Clk} \setminus A$ in B follows: given z, if $p(z)$ halts then search for a clockable x satisfying the right-hand side of (3.1), which is for each clockable x computable in B. Halt if such an x is found. (This procedure does not halt if and only if either $p(z)$ does not halt—which is equivalent with $z \notin \mathsf{Clk}$—or $z \in \mathsf{Clk}$ and no such x is found, from which $z \in A$ follows.)

Since A itself is semi-decidable, it is trivially semi-decidable in B. So we may from the above conclude $A \leqslant_{\mathrm{ITTM}} B$.

To finish the proof, we must show that $B \leqslant_{\mathrm{ITTM}} A$. A reduction machine works as follows: given a real z, it reject if it does not code an ordinal. If it does code an ordinal α, it can use this code to obtain—not necessarily canonical—codes for all $\beta, \gamma \leq \alpha$, and can thus search for codes for $\beta, \gamma \leq \alpha$ so that the order type of $\langle \beta, \gamma \rangle$, for which a not necessarily canonical code can be computed in terms of codes for β, γ (lemma A.7), is α. Codes for such $\beta, \gamma \leq \alpha$ will always be found since Cantor pairing is bijective and $\max \{\varepsilon, \delta\} > \alpha \to \langle \varepsilon, \delta \rangle > \langle \alpha, \alpha \rangle \geq \alpha$. Once computed, the machine simulates all $\{e\}(0)$ for β steps and for γ steps to check if β, γ are clockable ordinals. If not, z is rejected. If β, γ are clockable, then the machine can find canonical codes w, x for β and γ. At this point, it is also clear that $\langle \beta, \gamma \rangle$ is writable, so its canonical code can safely be computed and if this is not z, then z is rejected. Finally, the machine has to check

$$(\exists y \in \mathsf{Clk})\, \gamma < y \wedge p(f(y)) < p(f(\gamma)) \wedge f(y) < \beta$$

which it can do as follows: it iterates over β while querying A to iterate over all $y' \in A \cap \beta$. For each such y', it finds the y with $f(y) = y'$ (which must exist) by generating Clk. It can then check $\gamma < y \wedge p(f(y)) < p(f(\gamma))$ since p, f are computable. If this holds for any y found, the machine accepts z, and otherwise it rejects (because all y with $f(y) < \alpha$ have been considered). □

We restate the following, based on the last reduction of the proof above.

Lemma 3.23. $\mathsf{Clk} \otimes \mathsf{Clk}$ *is decidable.*

There are many cofinal semi-decidable subsets of Clk.

Lemma 3.24. *If $A \subseteq B \subseteq \mathsf{Won}$ and B is decidable, then A is equivalent with some $A' \subseteq \mathsf{Clk}$. Furthermore, if A is regular, A' is too.*

Proof. Let $f : \mathsf{Clk} \to \mathsf{Won}$ be a computable, order-preserving bijection, which exists by lemma 3.20. Define $A' = f^{-1}[A]$. We have $A' \leqslant_{\mathrm{ITTM}} A$ by the decidability of Clk and the computability of f: given x, first check if $x \in \mathsf{Clk}$ and if so, whether $f(x) \in A$.

If $\beta \in \mathsf{Won}$ and $A \cap \beta$ is decidable, then $A' \cap f^{-1}(\beta) = f^{-1}[A \cap \beta]$ reduces to $A \cap \beta$ by a similar procedure, and hence is decidable as well. Hence, by lemma 3.12, if A is regular then A' is too.

Conversely, to reduce A to A': given x, reject if $x \notin B$. Otherwise, x is a canonical code for a writable ordinal, and hence $f^{-1}(x)$ exists and can be found by generating Clk. Accept if and only if $f^{-1}(x) \in A'$. □

Lemma 3.25. *If $A \subseteq \mathsf{Clk}$ then it is equivalent with a cofinal $A^* \subseteq \mathsf{Clk}$. Furthermore, if A is regular, then A^* is too.*

Proof. By the above, we may assume without loss of generality that $A \subseteq \mathsf{Clk}$. If A is cofinal then there is nothing to do. So assume A is not.

Then there is an $\alpha \in \mathsf{Won}$ so that $A \cap \alpha = A$. Define $A^* = A \cup (\mathsf{Clk} \setminus (\alpha + 1))$. Since α is writable and Clk is decidable, $\mathsf{Clk} \setminus (\alpha + 1)$ is decidable. Hence A reduces to A^*: accept x if and only if $x \in A^*$ and $x \notin \mathsf{Clk} \setminus (\alpha + 1)$.

Likewise, A^* can be reduced to A by, given x, accepting if and only if $x \in A$ or $x \in \mathsf{Clk} \setminus (\alpha + 1)$.

Finally, if A is regular then A^* is too, since $(\mathsf{Clk} \setminus (\alpha + 1))$ is regular and a union of two decidable sets is decidable. □

Corollary 3.26. *If $A \subseteq B \subseteq \mathsf{Won}$ and B is decidable, then A is equivalent with some cofinal $A^* \subseteq \mathsf{Clk}$. Furthermore, if A is regular, then A^* is too.*

Corollary 3.27. *If $A \subseteq B \subseteq \mathsf{Won}$ is semi-decidable and B is decidable, then A is equivalent with a regular $C \subseteq \mathsf{Clk}$.*

Proof. Combining the lemmas above: A is equivalent with a cofinal $A^* \subseteq \mathsf{Clk}$, which is equivalent with a regular $D \subseteq \mathsf{Clk} \otimes \mathsf{Clk}$, which by the decidability of $\mathsf{Clk} \otimes \mathsf{Clk}$ is equivalent with a regular $D^* \subseteq \mathsf{Clk}$. □

We now turn to simple sets.

Lemma 3.28. *If X is ITTM-finite and there is a computable injection f and a decidable $B \subseteq \operatorname{dom} f \cap \mathsf{Won}$ with $X \subseteq f[B]$, then $Y = f^{-1}[X]$ is ITTM-finite and $Y \equiv_{\mathrm{ITTM}} X$.*

Proof. Since X is ITTM-finite, an ITTM can iterate over its contents, clock for every element $x \in X$ the unique $y \in B \subseteq \mathsf{Won}$ with $f(y) = x$, and then halt. It follows that $\beta = \sup Y < \Gamma$, so it suffices to show $Y \equiv_{\mathrm{ITTM}} X$.

X reduces to Y by, given a real z, iterating over all elements b of B that are ordinals below β (formally, iterating over all predecessors b of β and checking if they are in the decidable B) and seeing if $f(b) = z$.

Y reduces to X by, given a real z, rejecting if $z \notin B$ and otherwise computing $f(z)$ and checking $f(z) \in X$. □

Lemma 3.29. *If $A \subseteq \mathsf{Clk}$ is regular and undecidable, it is equivalent with a regular, simple set.*

Proof. We mostly follow the proof of the original theorem 3.6. Let A be regular and undecidable. All $A \cap \beta, \beta \in \mathsf{Won}$ are decidable by the regularity of A, so by its undecidability A must be cofinal in Won. Hence (corollary 3.21) there is a computable bijection $f : \mathsf{Clk} \to A$. Define

$$S = \{\alpha \in \mathsf{Clk} : (\exists \beta \in \mathsf{Clk})\, \beta > \alpha \wedge f(\beta) < f(\alpha)\}$$

which is semi-decidable since Clk can be generated.

That S is regular follows from the observation (cf. [18, lemma 4.1]) that for all $\delta \in \mathsf{Won}$, $g(\delta) = f^{-1}(\min f[\mathsf{Clk} \setminus \delta])$ is writable and

$$\alpha \in S \cap \delta \leftrightarrow \alpha \in \mathsf{Clk} \wedge \alpha < \delta \wedge (\exists \beta \in \mathsf{Clk})\, \alpha < \beta \leq g(\delta) \wedge f(\beta) < f(\alpha)$$

holds: if there is a $\beta > g(\delta)$ with $f(\beta) < f(\alpha)$, then $\beta = g(\delta)$ suffices as well since $(\forall \beta' > g(\delta))\, f(\beta') > f(g(\delta))$.

S reduces to A since

$$\alpha \in S \leftrightarrow \alpha \in \mathsf{Clk} \wedge (f(\alpha) \cap A \setminus f[\alpha]) \neq \emptyset,$$

and Clk, $f(\alpha)$, $f[\alpha]$ are all decidable for clockable α. Given α, the $\beta > \alpha$ with the smallest $f(\beta)$ lies outside S, so S is cocofinal.

If $\alpha \in \mathsf{Clk} \setminus S$ and $\beta < f(\alpha)$, then $\beta \in A \leftrightarrow \beta \in f[\alpha]$. Such an α can be found for every β: $f[\mathsf{Clk} \setminus S]$ is cofinal in Clk since $A = f[\mathsf{Clk}]$ is regular and $\mathsf{Clk} \setminus S$ is cofinal. (If $X = f[\mathsf{Clk} \setminus S] \subseteq A$ were bounded, then it were ITTM-finite by the regularity of A, and by the lemma above $\mathsf{Clk} \setminus S$ would be ITTM-finite.) It follows that A reduces to $\mathsf{Clk} \setminus S$, and thus to S.

Similarly, if $\mathsf{Clk} \setminus S$ were to contain a cofinal semi-decidable set Y', then $X = f[Y']$ would have to be cofinal in Clk as well: otherwise, by regularity of A, $Y = A \cap (\sup Y' + 1)$ would be ITTM-finite, and thus $f^{-1}[Y] \supseteq X$ would be ITTM-finite. But then one could, as above, find a suitable α for every β by generating Y', making A decidable. So $\mathsf{Clk} \setminus S$ contains no cofinal semi-decidable sets. $\square$

Corollary 3.30. *If $A \subseteq B \subseteq \mathsf{Won}$, A is undecidable and semi-decidable, and B is decidable, then A is equivalent with a regular simple set.*

So, as $h = \{e : \{e\}(0)\downarrow\} \subseteq \mathsf{Clk}$ is undecidable and semi-decidable, we find:

Corollary 3.31. *The ITTM-semi-decidable degrees contain a complete infinite binary tree. In particular, there are infinitely many semi-decidable degrees.*

3.3.4 Outside of Won

The above results can be transferred to well-behaved sets of writable reals.

Lemma 3.32. *If $X \subseteq Y \subseteq \mathsf{Wrr}$ is semi-decidable and Y is decidable, then X is equivalent with an $A \subseteq \omega \otimes \mathsf{Clk} \subseteq \mathsf{Clk} \otimes \mathsf{Clk}$.*

Proof. Define[9]

$$A = \{\langle e, \alpha \rangle : e \in \omega \wedge \alpha \in \mathsf{Clk} \wedge (\exists x \in X)\, \{e\}_\alpha(0) = x\}.$$

Since $\mathsf{Clk} \otimes \mathsf{Clk}$ is decidable and hence $\omega \otimes \mathsf{Clk}$ is decidable (ω is writable), C reduces to X by rejecting inputs outside of $\omega \otimes \mathsf{Clk}$, and on inputs $\langle e, \alpha \rangle$ with e, α canonical codes accepting if $\{e\}_\alpha(0) \in X$ and otherwise rejecting.

For $X \leqslant_{\text{ITTM}} A$: given a real x, reject if it lies outside Y. Otherwise, x is a writable real, and thus an $e < \omega$ and $\alpha \in \mathsf{Clk}$ with $\{e\}_\alpha(0) = x$ must exist and therefore can be computed. Accept if and only if $\langle e, \alpha \rangle \in A$. $\square$

From lemma 3.27 and lemma 3.30 we thus obtain the following.

[9]Ordering the pairs $\langle e, \alpha \rangle$ first on the first coordinate and then on the second, one can also find a set $A \equiv_{\text{ITTM}} X$ so that for every $x \in X$ there is exactly one pair $\langle e, \alpha \rangle \in A$ with $\{e\}_\alpha(0) = x$.

Corollary 3.33. *If $A \subseteq B \subseteq \mathsf{Wrr}$ be semi-decidable and B decidable, then A is equivalent with a regular set. If A is undecidable as well, then it is equivalent with a regular simple set.*

Corollary 3.34. *If $X \subseteq Y \subseteq \mathsf{Wrr}$ is undecidable and semi-decidable, and Y is decidable, then there are $P, Q \leqslant_{\mathrm{ITTM}} X$ so that $X, P \nleqslant_{\mathrm{ITTM}} Q$ and $X, Q \nleqslant_{\mathrm{ITTM}} P$.*

From the splitting theorem for TMs, one can conclude that there is no minimal TM-semi-decidable degree, apart from the degree of the TM-decidable sets. The same cannot be concluded here.

Question 1. *Are there non-trivial minimal ITTM-semi-decidable degrees?*

To answer this question negatively, it would suffice for there to exist for every undecidable semi-decidable $Z \subseteq \mathsf{Wrr}$ a semi-decidable $X \subseteq Y \subseteq \mathsf{Wrr}$ so that Y is decidable and $Z \equiv_{\mathrm{ITTM}} X$, though we would not expect this to be true, as a.o. Wrr is undecidable (corollary A.6).

3.4 For OTMs: an open question

Can a splitting theorem be proved for OTMs? We do not know. A variant of Dekker's theorem can be proved.

Definition 3.35. A class is OTM-simple if it is cocofinal in On and intersects all unbounded OTM-semi-decidable classes.

Lemma 3.36. *Every undecidable OTM-semi-decidable class is equivalent with an OTM-simple class.*

Proof sketch. Let A be undecidable and OTM-semi-decidable. By [10, lemma 4.2] and [10, theorem 6.2], there is (up to coding) an OTM-computable bijective $G : \mathsf{On} \to L$.

As in the proof of [18, theorem VII.4.2], define $A^* = \{\alpha : G(\alpha) \cap A \neq \emptyset\}$. We have $A^* \equiv_{\mathrm{OTM}} A$. Since A is nonempty, there are unboundedly many definable sets that intersect A, hence A^* is cofinal.

So we may assume without loss of generality that A is cofinal. Then there is a computable bijective $F : \mathsf{On} \to A$ by a similar proof as for lemma 3.19. Now define

$$S = \{\alpha : (\exists \beta > \alpha)\, F(\beta) < F(\alpha)\}$$

and follow the proof of Dekker's theorem for TMs. It is OTM-semi-decidable since OTMs can iterate over the ordinals. It is cocofinal since for every α, the $\beta > \alpha$ with least $F(\beta)$ lies outside S. Since $\alpha \in S \leftrightarrow A \setminus F[\alpha] \neq \emptyset$ holds, S reduces to A.

Assuming $\alpha \notin S$ and $\beta < F(\alpha)$, we have $\beta \in A \leftrightarrow \beta \in F[\alpha]$. Since $\overline{S}$ is unbounded, F is bijective and A, F are Σ_1-definable ("there exists a computation such that…"), $F[\overline{S}]$ must be unbounded: if it had a bound δ, then $\delta \cap A = \{\alpha < \beta : F(\alpha)\downarrow\}$ and consequently $\{\alpha \in A \cap \delta : \alpha \in F^{-1}[\overline{S}]\} \supseteq \overline{S}$ would be an unbounded set. Hence, A reduces to S by finding for every β an $\alpha \notin S$ with $\beta < F(\alpha)$.

If there were an unbounded OTM-semi-decidable X that did not intersect S, then, by similar reasoning as above, $F[X]$ would be unbounded, and A could be decided by, given β, finding an $\alpha \in X$ with $\beta < F(\alpha)$ by generating X. □

However, did not find a way to prove a bounded injury lemma. In the proofs so far, to see that all W_i are bounded, the simplicity of D is used: if $W_i \cap D$ is empty, then W_i is a subset of $\overline{D}$, which, because D is simple, cannot contain any unbounded semi-decidable subsets.

The usual proof that W_i is semi-decidable—assuming by induction that the W_j, $j < i$ have settled in some stage α, then finding a stage $\beta > \alpha$ in which all elements of C below the restraints of the W_j have entered $A_\beta \cup B_\beta$, making W_i semi-decidable in the parameter β—does not work, as OTMs are parameter-free machines: only countably many ordinals are "OTM-writable".[10]

3.5 For p-OTMs

For ordinal machines with parameters, the issues discussed for OTMs above evaporate. We sketch a proof. From hereon, computable, semi-decidable etc. are all understood to imply the p-OTM-variant of the definitions.

Definition 3.37. A class of ordinals is *p-OTM-finite* if it is bounded and p-OTM-decidable. Equivalently, by [12, theorem 12], if it is an element of L.

Definition 3.38. A class is *p-OTM-simple* if it is semi-decidable, cocofinal in the ordinals, and intersects every unbounded semi-decidable class.

For every p-OTM-semi-decidable class A there is a partial p-OTM-computable surjective and injective $F : \mathsf{On} \to A$ whose domain is either On or an ordinal α. This is constructed in a similar way as in lemma 3.19[11] (and mirrors the situation in computability theory, where semi-decidability is equivalent with being computably enumerable). Furthermore, if A is undecidable, then A must be unbounded: it if had a bound β then it would be an element of L, since A is defined in terms of a machine ε (cf. section 1.5). In particular, all simple classes are unbounded.

[10]Call an ordinal α OTM-writable if there is an OTM that, started on input 0, halts with only a 1 in the αth cell of the (output) tape. There are countably many machines, the OTM-writable ordinals form a set and are hence bounded. They are closed under addition, multiplication, and exponentiation. It might be interesting to attempt to prove some splitting theorem within the OTM-writable ordinals,

[11]Here, it is surjective because every approximation $\{\alpha < \beta : \{\varepsilon\}_\beta(\alpha)\downarrow\}$ is bounded.

Lemma 3.39 (Dekker for p-OTMs). *Every undecidable semi-decidable class is equivalent with a simple class.*

Proof sketch. By the above, A is unbounded, and there is a computable bijective class function $F : \mathsf{On} \to A$. Define the semi-decidable class S by

$$\alpha \in S \leftrightarrow (\exists \beta > \alpha)\, F(\beta) < F(\alpha).$$

S is unbounded and reduces to A by the usual Dekker argument.

If $\alpha \notin S$ and $\beta < F(\alpha)$, then $\beta \in A \leftrightarrow \beta \in F[\alpha]$. Since $\overline{S}$ is unbounded (and F is Σ_1-definable with parameters over L), $F[\overline{S}]$ must be a proper unbounded class, and A reduces to S by the same reasoning as in the proof of theorem 3.6. Similarly, if $\overline{S}$ were to contain an unbounded semi-decidable class then A could be computed by enumerating that class, contradicting that A is undecidable. □

Theorem 3.40. *If C is semi-decidable and D is OTM-simple with parameters, then C is the union of disjoint, semi-decidable A, B which D does not reduce to.*

Proof. The requirements are

$$\{\varepsilon\}^A \neq D \tag{R^A_ε}$$
$$\{\varepsilon\}^B \neq D \tag{R^B_ε}$$
$$\varepsilon \in C \leftrightarrow \varepsilon \in A \cup B \tag{R^C_ε}$$

and the $(R^A_\varepsilon), (R^B_\varepsilon)$ are computably ordered $\{R_\varepsilon : \varepsilon \in \mathsf{On}\}$. The construction of A_α, B_α is done as in algorithm 6, with the only difference that in stage σ only requirements $\{R_i : i < \sigma\}$ are considered.

Lemma 3.41 (Bounded injury). *All requirements settle: for each R_i there is an α such that R_i is never injured after stage α, and $W_i = \bigcup_{\alpha<\beta} W_{i,\beta}$ is p-OTM-finite.*

Proof. Note that if W_i exists, then it is definable by the formula $\Phi(i, w)$ given by

$$(\exists \alpha)(\forall \beta > \alpha)\, P_i(\beta) \wedge w \in W_{i,\beta}$$

where $P_i(\beta)$ is a formula describing that R_i is not injured in stage β, which can be written in terms of machines enumerating C, D and a formal description of the construction, and $W_{i,\beta}$ can be defined in those parameters as well. Similarly, if the stage $\alpha(i)$ in which R_i settles exists, it is defined by a formula with parameters.

Hence, by induction: if all $R_\beta, \beta < i$ settle, then they have all settled at stage $\alpha = \sup\{\gamma : (\exists \beta < i)\, \gamma = \alpha(\beta)\}$. Now conclude as before (the collection of first appearances of elements in the segment of C below the r_i is a set and hence bounded) that there is a stage beyond stage α after which R_i is never injured, hence W_i must exist, and it must be bounded (lest it is a cofinal semi-decidable subclass of $\overline{D}$) and so an element of L, since it is bounded and semi-decidable. (As seen before, all undecidable semi-decidables are unbounded.) □

Lemma 3.42 (Reflection). *If $\{e\}^A(w) = 0$ then there is some α with $\{e\}_\alpha^{A_\alpha}(w) = 0$, and similarly for B.*

Proof. Consider the positive queries Q made in $\{e\}^A(w) = 0$. Because A is semi-decidable and the computation has an ordinal length, putting a bound on the queries, Q is a set. Then the collection of individual first stages of appearance of the elements of Q in A is a set as well, and so it has a supremum. □

Now the proof can be completed as before. □

Corollary 3.43. *If C is a semi-decidable degree that is not the degree of decidable classes, it can be split into two incomparable semi-decidable degrees whose least upper bound is C.*

Corollary 3.44. *The class of semi-decidable degrees contains a complete binary tree.*

Corollary 3.45. *The class of semi-decidable degrees without the degree of the decidable classes has no minimum.*

There is a proof for the splitting theorem in α-recursion theory by Shore [20] which we believe can be adapted to p-α-machines as they are a machine model for α-recursion theory, but we have not checked. In the case of α-machines, we do not know whether a splitting theorem holds, cf. section 3.4.

Chapter 4

The thickness lemma

The priority arguments discussed thus far are known as finite injury arguments, because (in the TM case) each requirement is proved to be injured at most finitely often. Infinite injury arguments do not involve a injury lemma. Instead, it is proved that for each requirement, the set of injuries sustained is of low degree.

In this chapter we discuss the thickness lemma, which is proved by an infinite injury argument. Soare proved [23, section 3] that a strengthened version of the thickness lemma (obtained by technical modifications of the proof below) implies other infinite injury results, among which the density theorem, stating that between two semi-decidable degrees there is another semi-decidable degree.

First, we discuss a proof of the thickness lemma for Turing machines. We then show that a naive translation to transfinite computability theory does not hold for any of the transfinite machine models discussed. A modified translation is then proved for certain ITTM-semi-decidable sets of low degree. Finally, we briefly sketch a thickness lemma for p-OTMs.

4.1 For Turing machines

The thickness lemma is generally attributed to Shoenfield, though the current presentation and the proof below are based on the work of Soare [23, section 2] and Odifreddi [14, section X.3].

Definition 4.1. $A \subseteq B$ is a *thick subset* of B if $B^{[e]} \setminus A^{[e]}$ is finite for all $e < \omega$.

Theorem 4.2 (Thickness lemma, Shoenfield)**.** *If B, C are semi-decidable, C is undecidable and does not reduce to $B^{[<e]}$ for any $e < \omega$, then there is a semi-decidable thick subset A of B that C does not reduce to.*

Making sure to only add elements of B to A, the requirements are

$$\{e\}^A \neq C \qquad (N_e)$$

$$B^{[e]} \setminus A^{[e]} \text{ is finite} \qquad (P_e)$$

prioritized $(N_0) < (P_0) < (N_1) < (P_1) < \dots$ so that (N_0) has highest priority.

The idea is for requirements (N_e) to find and try to preserve a computation $\{e\}_s^{A_s}(w) = 0$ with $w \in C_s$, and for requirements (P_e) to add to A_s all elements of $B_s^{[e]}$ unrestrained by the $(N_{e'})$ of higher priority.

Because B is generally undecidable and its columns infinite, the number of stages in which new elements of $B^{[e]}$ are discovered can be infinite, and hence it is generally impossible to keep the injuries finite. This is where the infinite injury argument comes in.

Simultaneously define

$$\begin{aligned} u(e,x,s) &= \begin{cases} \sup\{y : y \text{ is queried in } \{e\}_s^{A_s}(x)\} & \text{if } \{e\}_s^{A_s}(x) \text{ halts;} \\ 0 & \text{otherwise.} \end{cases} \\ l(e,s) &= \sup\{x < s : (\forall y < x)\, C_s(y) = \{e\}_s^{A_s}(y)\}; \\ r(e,s) &= \sup\{u(e,x,s) + 1 : x \le l(e,s)\}; \end{aligned}$$

respectively the *use*, *length of agreement*, and *restraint* functions. In the above, C_s is identified with the characteristic function of C_s, an s-step approximation of C. The approximations A_s of A depend on the functions just defined, so they are defined recursively in terms of each other. Likewise, define

$$a_s = \begin{cases} \min(A_s \setminus A_{s-1}) & \text{if } A_s \setminus A_{s-1} \neq 0; \\ \sup A_s \cup s & \text{otherwise;} \end{cases}$$

the recursively defined machine transformation $\hat{\cdot}$ by

$$\{\hat{e}\}_s^{A_s}(x) = \begin{cases} \{e\}_s^{A_s}(x) & \text{if } u(e,x,s) < a_s; \\ \uparrow & \text{otherwise;} \end{cases}$$

and finally

$$\hat{u}(e,x,s) = u(\hat{e},x,s); \qquad \hat{l}(e,s) = l(\hat{e},s); \qquad \hat{r}(e,s) = r(\hat{e},s).$$

The approximations A_s of A are then defined by

$$\begin{aligned} A_0 &= \varnothing; \\ A_{s+1} &= A_s \cup \bigcup_{e<s} B_{s+1}^{[e]} \setminus \hat{R}(e+1,s), \end{aligned} \tag{4.1}$$

with $\hat{R}(e,s) = \sup_{d<e} \hat{r}(d,s)$. The idea is that the restraint put on (P_e) is given by $\hat{R}(e+1,s)$, the restraints of the higher priority $(N_{e'}), e' < e$. The individual restraints $r(e,s)$ themselves aim to preserve the computations $\{e\}_s^{A_s}(y)$ for all $y \le l(e,s)$, which includes the first point where C_s and $\{e\}_s^{A_s}$ disagree, if there is one. Soare's "hat trick" is used to make restraints drop frequently,[12] providing windows of opportunity for the (P_e) to add elements to A.

We leave it to the reader to prove that the definitions above lead to a stages-based algorithm similar to algorithm 1.

Definition 4.3. Stage $s < \omega$ is a *true stage* if $A_s \cap a_s = A \cap a_s$.

Such stages are called true because all halting computation are preserved: if $\{\hat{e}\}_s^{A_s}(x)$ halts for some true stage s, then for all true $s' > s$ we have (since necessarily $a_{s'} > a_s$) that $\{\hat{e}\}_s^{A_s}(x) = \{\hat{e}\}_{s'}^{A_{s'}}(x) = \{e\}^A(x)$ holds. Furthermore, every halting computation $\{e\}^A(x)$ is reflected in all true stages s for which a_s is greater than the positive queries made in $\{e\}^A(x)$. Since $\{a_s : s \in \omega\}$ is cofinal in ω, such stages always exist.

Lemma 4.4. *The set T of true stages is cofinal in ω.*

Proof. For any s, the first stage $t > s$ with the least a_t satisfies $A_t \cap a_t = A \cap a_t$. $\square$

The rest of the proof relies on two key lemmas. Define the *injury sets* and their approximations

$$\begin{aligned}\hat{I}_e &= \{x : (\exists s)\, x < \hat{r}(e,s) \wedge x \in A_{s+1} \setminus A_s\} \\ \hat{I}_{e,s} &= \{x : (\exists t \leq s)\, x < \hat{r}(e,t) \wedge x \in A_{t+1} \setminus A_t\}.\end{aligned} \tag{4.2}$$

For each e, the set $\hat{I}_e$ contains the elements x that are added to A while being restrained by (N_e), so these are the elements that injure (N_e).

The rest of the proof relies on two key lemmas.

Lemma 4.5 (Injury lemma, Soare). *If $C \not\leq_{\mathrm{TM}} \hat{I}_e$ then $C \neq \{e\}^A$.*

Lemma 4.6 (Window lemma, Soare). *If $C \neq \{e\}^A$ then $\lim_{t \in T} \hat{r}(e,t) < \omega$ exists.*

The injury lemma provides a way to prove that requirements (N_e) are satisfied, and the window lemma states that the satisfaction of all $(N_{e'}), e' \leq e$ implies the satisfaction of (P_e): if $\hat{R}_e = \liminf_{s<\omega} \hat{R}(e,s)$ exists, all elements of $B^{[e]}$ above $\hat{R}_e$ are eventually added to A, making $B^{[e]} \setminus A^{[e]} = \hat{R}_e$ finite.

Proof of the injury lemma. We prove the contrapositive. Assume $C = \{e\}^A$. Then $\lim_{s<\omega} l(e,s) = \omega$ and C can be reduced to $\hat{I}_e$ as follows. Given x, it suffices to find a stage s such that $\{e\}_s^{A_s}(x) = \{e\}^A(x)$. (That s exists follows from an observation similar to the reflection lemmas seen before.) Such states are characterized by

$$x < l(e,s) \wedge \{e\}_s^{A_s}(x)\downarrow \wedge (\forall y \leq u(e,x,s))\, y \notin \hat{I}_e \setminus A_s$$

[12]Frequently, in the sense that the hat trick—so aptly named by Odifreddi [14, p. 478]—makes a requirement drop its restraint if an element below its restraint is added to A, even if this added element does not impact any of the actual computations it is trying to preserve (even if the length of agreement does not drop).

since then the computation $\{e\}_s^{A_s}(x)$ halts and will not be injured in stages $t > s$. So C can be reduced to $\hat{I}_e$. □

The proof of the window lemma relies on the true stages.

Proof of window lemma. Assume $C \neq \{e\}^A$. Let x be the least with $C(x) \neq \{e\}^A$ and let t be a true stage in which $\{\hat{e}\}_t^{A_t}(y) = \{e\}^A(y)$ holds for all $y < x$. (For example, the first true stage after the first stage in which all positive queries made in the halting computations $\{e\}^A(y)$ have entered A.)

As noted before, all true stages $t' > t$ must then preserve these computations: $\{\hat{e}\}_{t'}^{A_t}(y) = \{\hat{e}\}_t^{A_t}(y) = \{e\}^A(y)$ holds for all $y < x$.

If $\{e\}^A(x)$ does not halt, then $\{\hat{e}\}^A(x)$ never halts in any true stage (by the preservation property), and so we find $\hat{r}(e, t') = \hat{r}(e, t)$ for all true $t' > t$.

If $\{e\}^A(x)$ does halt, then find similarly a true stage $s > t$ which reflects this. By same reasoning as above, $\{\hat{e}\}^A(x + 1)$ never halts in any true stage, and thus $\hat{r}(e, s) = \hat{r}(e, s')$ for all true $s' > s$. □

Proof of the thickness lemma. Let B, C be semi-decidable, such that C is incomputable and does not reduce to $B^{[<e]}$ for any $e < \omega$. We show by induction that $A = \cup_{s<\omega} A_s$ defined as in (4.1) satisfies all requirements.

Assume all requirements below (N_e) are satisfied. Then in particular, for all $e' < e$, $(P_{e'})$ is satisfied, $B^{[e']} \setminus A^{[e']}$ is finite, and thus $B^{[<e]} \setminus A^{[<e]}$ is finite. It follows (because of $B^{[<e]} \subseteq A^{[<e]}$ and the finite difference, which can be coded in a reduction machine) that $A^{[<e]}$ reduces to $B^{[<e]}$. (N_e) can only be injured by the $(P_{e'})$, which only add elements to the eth section of A, and so $\hat{I}_e \subseteq A^{[<e]}$ holds. It follows that $\hat{I}_e$ reduces to $A^{[<e]}$: given x, check whether $x \in A^{[<e]}$ and if so, run the construction until x is added to some A_s and check whether it injures (N_e) in stage s, cf. (4.2). Hence $\hat{I}_e \leqslant_{\mathrm{TM}} B^{[<e]}$, and the injury lemma together with $C \not\leqslant_{\mathrm{TM}} B^{[<e]}$ proves that (N_e) is satisfied.

Now assume that all requirements below (P_e) are satisfied. Then for all $e' \leq e$ we have that $(N_{e'})$ is satisfied, and by the window lemma $\lim_{t \in T} \hat{r}(e', t)$ exists. Thus (since T is unbounded) we find $\liminf_{s<\omega} \hat{R}(e + 1, s) < \omega$, from which it follows that (P_e) is satisfied. □

By altering the definition of $\hat{I}$, stronger versions can be obtained, such as

Theorem 4.7 (Strong thickness lemma, Shoenfield, [14, theorem X.3.9]). *If B, C are semi-decidable, C is undecidable and does not reduce to $B^{[<e]}$ for any $e < \omega$, then there is a thick subset $A \leqslant_{\mathrm{TM}} B$ of B that C does not reduce to.*

4.2 Transfinite thickness lemmas

Classically, the thickness lemma shows that one can find a subset $A \subseteq B$ where all column differences are finite. In the proof above, this partly hinges on the set of queries made in computations being finite, from which it follows that $\hat{R}(e, s)$ is always finite. This fails in the transfinite setting.

So the classical proof of the thickness lemma cannot be used to prove a thickness lemma for transfinite computability, where column differences are kept finite. In fact, no such proof exists.

Theorem 4.8 (Failure of the transfinite thickness lemma with finite differences)**.** *There are ITTM-semi-decidable sets B, C such that C is un-ITTTM-decidable, does not ITTM-reduce to any $B^{[<e]}$, $e < \omega$, and yet ITTM-reduces to all thick subsets $A \subseteq B$.*

Proof. Let $C = \{e : \{e\}(0)\downarrow\}$ and define $B = C \otimes \omega$ (writing ω for the set of canonical codes for natural numbers). That B is semi-decidable follows from the fact that C is (proposition 1.28) and that ω can be iterated over. Since C is undecidable (proposition 1.28), and $B^{[<e]}, e < \omega$ reduces to the finite set $C \cap e$ and is thus decidable (its contents can be encoded in the code of a machine), $C \nleq_{\text{ITTM}} B^{[<e]}$ holds for all $e < \omega$.

Let A be a thick subset of B. Then, by definition of B, we have

$$e \in C \leftrightarrow (\exists n < \omega)\, \langle e, n\rangle \in B$$

since $B^{[e]} \setminus A^{[e]}$ is finite for all e. Hence C reduces to B by, given x, checking whether x is a canonical code a natural number and if so, iterating over all $n < \omega$ and checking $\langle x, n\rangle \in B$. □

In fact, for any semi-decidable C such that $(\forall e < \omega)\, C \nleq_{\text{ITTM}} C \cap e$ holds, C reduces to all thick subsets of the semi-decidable $C \otimes \omega$. Similar arguments can be made for the other transfinite machine models discussed.

Corollary 4.9. *Let M be a transfinite machine model discussed (not ω-machines). There are M-semi-decidable sets B, C such that C is not M-decidable, does not M-reduce to any $B^{[<e]}$, $e < \omega$, yet M-reduces to all thick $A \subseteq B$.*

However, if we instead look for subsets with small but not necessarily finite column differences, some generalized results can be obtained. Recall that the classical proof relies on alternating[13] positive requirements (P_e) with negative requirements (N_e), where each positive requirement corresponds to a column and each negative requirement to a machine. So, in order to adapt this to parametrized machine models, the number of columns has to be increased to match the number of machines. For ITTMs, the number of columns can instead be reduced to the number of machines.

[13]Or, at least, that both types of requirements are cofinal in the priority order.

4.3 For ITTMs

We prove a thickness lemma for semi-decidable subsets of Won of low degree, where instead of finite differences we consider ITTM-finite differences. In this section, the pairing function $\langle\cdot,\cdot\rangle$ refers to the Cantor pairing function on ordinals. This is ITTM-computable (lemma A.7), so, in particular, pairings of writable ordinals are writable, hence $\Gamma = \Gamma \otimes \Gamma$. In this section, we write semi-decidable for ITTM-semi-decidable, et cetera.

Definition 4.10. Call $A \subseteq B$ an *ITTM-thick subset* if the differences $B^{[\alpha]} \setminus A^{[\alpha]}$ are ITTM-finite for all $\alpha < \Gamma$.

In the classical proof, we use (in the proof of the window lemma) that sets of queries made in halting computations $\{e\}^A(x)$ are bounded. This does not generalize to the current situation, as queries made against the weak halting problem $h = \{e : \{e\}(0)\downarrow\}$ can be unbounded in Won: a machine could query h to see which machines m halt, then simulate all halting computations $\{m\}(0)$, query their outputs, and then halt.

This proves

Lemma 4.11. *If $h \leqslant_{\text{ITTM}} A$ then there is a machine e and a $w \in$ Won so that $\{e\}^A(w)$ halts and the set of oracle queries made is unbounded in* Won.

Conversely:

Lemma 4.12. *If $A \subseteq$ Won is semi-decidable and there is a machine e and a $w \in$ Won so that $\{e\}^A(w)$ halts and the set of oracle queries made is unbounded in* Won*, then $A \leqslant_{\text{ITTM}} h$.*

Proof. The reduction works as follows: given x, reject if is not a canonical code for some $e < \omega$ (by generating Clk up to ω, for example). Otherwise, simulate $\{m\}^A(w)$, which can be done in the oracle h since, by proposition 1.29, $A \leqslant_{\text{ITTM}} h$. For every ordinal query α made in $\{m\}^A(w)$, simulate $\{e\}_\alpha(0)$. Accept x if $\{e\}_\alpha(0)$ halts for any α queried. Otherwise—if the simulation of $\{m\}^A(w)\downarrow$ has concluded $\{e\}_\alpha(0)$ has not halted for any α queried—reject x. □

Combining the above with proposition 1.29, we find

Corollary 4.13. *Let $A \subseteq$ Won be semi-decidable. There are machines e and $w \in$ Won so that $\{e\}^A(w)$ halts and the set of oracle queries made is unbounded in* Won *if and only if $A \equiv_{\text{ITTM}} h$.*

Corollary 4.14. *Let $A \subseteq$ Won be semi-decidable with $A <_{\text{ITTM}} h$ and assume that $\{e\}^A(v)$ halts for all $v \leq w < \Gamma$. Then the total set of oracle queries made in the computations $\{e\}^A(v), v \leq w$ is bounded in* Won.

Proof. Since w is writable and Won is generable in-order, there is an ITTM which, on input 0 and equipped with the oracle A, simulates $\{e\}^A(v)$ for all $v \leq w$ and then halts. The result then follows from corollary 4.13. □

This observation allows us to prove the following.

Theorem 4.15. *Let $B, C \subseteq$ Won be semi-decidable so that C is contained in a decidable C' and $h \not\leq_{\mathrm{ITTM}} B$ (equivalently, $B <_{\mathrm{ITTM}} h$). If C does not reduce to any finite set of columns of B, then there is an ITTM-thick semi-decidable $A \subseteq B$ that C does not reduce to.*

The assumption that C does not reduce to any finite set of columns of B is implied by C not reducing to any $B^{[<\alpha]}$, $\alpha \in$ Won.

Proof. There are requirements

$$\{e\}^A \neq C \tag{N_e}$$

$$B^{[\alpha]} \setminus A^{[\alpha]} \text{ is ITTM-finite} \tag{P'_α}$$

for each $e < \omega$ and $\alpha \in$ Won. To use a similar argument structure as before, the (N_e)-requirements would have to be cofinal in the priority ordering. We use an injective computable projection function $p :$ Won $\to \omega$ (for example, sending ordinals to machines that write them) to reduce the requirements to

$$\{e\}^A \neq C \tag{N_e}$$

$$\text{if } \alpha = p^{-1}(e) \text{ exists, then } B^{[\alpha]} \setminus A^{[\alpha]} \text{ is ITTM-finite} \tag{P_e}$$

for all $e < \omega$, ordered $(N_0) < (P_0) < (N_1) < \ldots$ as before. Again, define

$$u(e,x,s) = \begin{cases} \sup\{y \in \mathrm{Won} : y \text{ is queried in } \{e\}_s^{A_s}(x)\} & \text{if } \{e\}_s^{A_s}(x) \text{ halts;} \\ 0 & \text{otherwise.} \end{cases}$$

$$l(e,s) = \sup\{x < s : (\forall y < x)\, C_s(y) = \{e\}_s^{A_s}(y)\};$$
$$r(e,s) = \sup\{u(e,x,s) + 1 : x \leq l(e,s)\};$$

and

$$a_s = \begin{cases} \min(A_s \setminus A_{<s}) & \text{if } A_s \setminus A_{<s} \neq 0; \\ \sup A_s \cup s & \text{otherwise.} \end{cases}$$

$$\{\hat{e}\}_s^{A_s}(x) = \begin{cases} \{e\}_s^{A_s}(x) & \text{if } u(e,x,s) < a_s; \\ \uparrow & \text{otherwise.} \end{cases}$$

$$\hat{u}(e,x,s) = u(\hat{e},x,s); \qquad \hat{l}(e,s) = l(\hat{e},s); \qquad \hat{r}(e,s) = r(\hat{e},s),$$

defining A by

$$\begin{aligned} A_0 &= \varnothing; \\ A_{s+1} &= A_s \cup \bigcup \left\{ B_{s+1}^{[\alpha]} \setminus \hat{R}(e+1,s) : e < s \wedge \alpha = \{p^\dagger\}_s(e) \right\}; \\ A_\lambda &= \bigcup_{s<\lambda} A_s \end{aligned} \tag{4.3}$$

where $\hat{R}(e,s) = \sup_{e'<e} \hat{r}(e',s)$ and $\{p^\dagger\}$ is a machine that, on input e, finds $p^{-1}(e)$ in clockable time if it exists, and otherwise never halts.[14] That the function

[14]For example, while generating Won, compute $p(\alpha)$ for each α found until $p(\alpha) = e$.

$s \mapsto A_s$ is computable by an ITTM (in particular, that A is thus ITTM-semi-decidable) is left to the reader.

The injury sets are defined similarly, where $\hat{I}_e = \bigcup_{s \in \mathsf{Won}} \hat{I}_{e,s}$ and

$$\hat{I}_{e,s} = \{x : (\exists t \leq s)\, x < \hat{r}(e,t) \wedge x \in A_{t+1} \setminus A_t\}.$$

By the same argument as for Turing machines, the true stages

$$T = \{s \in \mathsf{Won} : A_s \cap a_s = A \cap a_s\}$$

are cofinal in Won and preserve halting computations.

Lemma 4.16 (Injury lemma). *If $C \not\leqslant_{\mathrm{ITTM}} \hat{I}_e$, then $C \neq \{e\}^A$.*

Proof. Assuming $C = \{e\}^A$, then $l(e,s)$ is unbounded in s and C can be reduced to $\hat{I}_e$ by, given some x, rejecting if $x \notin C'$. Otherwise, $x \in \mathsf{Won}$, and an ITTM can find a state $s \in \mathsf{Won}$ with $\{\hat{e}\}_s^{A_s}(x) = \{e\}^A(x)$ (which exists by a runtime reflection argument as in lemma 2.9) as follows. Generate Won and, using the oracle $\hat{I}_e$ search for a stage with

$$x < \hat{l}(e,s) \wedge \{\hat{e}\}_s^{A_s}(x)\downarrow \wedge (\forall y \leq u(e,x,s))\, y \notin \hat{I}_e \setminus A_s.$$

Then $\{\hat{e}\}_s^{A_s}(x)\downarrow$ and the computation will be preserved, since if any element queried negatively in the computation were to be added to A later, it would injure (N_e) and thus be an element of $\hat{I}_e \setminus A_s$. So $\{\hat{e}\}_s^{A_s}(x) = \{e\}^A(x) = C(x)$, and x is accepted if and only if $\{\hat{e}\}^{A_s}(x) = 1$. □

For the window lemma, a new strategy is needed. Define

$$A^e = \bigcup_{\alpha \in p^{-1}[e]} A^{[\alpha]}$$

so that $\hat{I}_e \subseteq A^e$.

Lemma 4.17 (Window lemma). *If $C \neq \{n\}^A$ for all $n < e$ and $A^e \leqslant_{\mathrm{ITTM}} B$, then* $\liminf_{s<\Gamma} \hat{R}(e+1,s) < \Gamma$ *and $A^{e+1} \leqslant_{\mathrm{ITTM}} B$.*

Proof. For each $n < e$, let x_n be the least with $\{e\}^A(x_n) \neq C(x_n)$, and let z_n be $x_n + 1$ if $\{e\}^A(x_n)\downarrow$ or x_n otherwise. There is an oracle machine that, given an oracle for B and using a reduction machine for $A^e \leqslant_{\mathrm{ITTM}} B$, finds the first stage α such that

- for all $n < e$ and all $y_n < z_n$, the computations $\{\hat{n}\}_\alpha^{A_\alpha}(y_n)$ halt, and
- for all elements q queried and all $q' \leq q$ it holds that $q' \notin A^e \setminus A_\alpha$.

By the choice of the z_n, we have $\alpha \leq \Gamma$. If $\alpha = \Gamma$ were to hold, then h could be reduced to A^e by, given m, simulating the oracle machine just described and simulating $\{m\}(0)$ for as many steps as that simulation takes. (Accept if this simulation halts, otherwise reject.) Since $A^e \leqslant_{\text{ITTM}} B$, this would contradict the assumption $h \nleqslant_{\text{ITTM}} B$ on B. So $\alpha < \Gamma$.

Because the computations $\{\hat{n}\}_{\alpha}^{A_\alpha}(y_n)$ are preserved after stage α (they are only be injured if elements in A^e below their queries are added to A, but by the choice of α this does not occur), we have $\hat{R}(e+1,\alpha) \leq \hat{R}(e+1,\beta)$ for all $\beta \geq \alpha$. Furthermore, since $\hat{l}(n,t) \leq \hat{l}(n,\alpha)$ for all $n < e$ and true stages $t \geq \alpha$ and because the true stages are cofinal, it follows that $\hat{R}(e+1,t) \leq \hat{R}(e+1,\alpha)$ and thus $\hat{R}(e+1,t) = \hat{R}(e+1,\alpha)$ for all true $t \geq \alpha$. Hence $\lim_{t\in T} \hat{R}(e+1,t) = \hat{R}(e+1,\alpha)$, and so (again by the cofinality of T) we find $\liminf_{s<\Gamma} \hat{R}(e+1,s) = \hat{R}(e+1,\alpha)$. As A_α is computable by design of the algorithm and the writability of α, it follows by corollary 4.14 that $\hat{R}(e+1,\alpha) < \Gamma$ holds as desired.

Finally, A^{e+1} can be reduced to B as follows: find α. Given x, reject if it does not code an ordinal or if it is not in B. Otherwise, x codes some $\eta = \langle \gamma, \delta \rangle < \Gamma$. If $p(\gamma > e)$, then reject. If $p(\gamma) < e$, accept x if and only if $x \in A^e$. If $p(\gamma) = e$, then $\eta \in A^{e+1}$ if and only if $\eta \in A_\alpha$ or $\eta \geq \hat{R}(e+1,\alpha)$. □

Remark 4.18. The idea to use reductions from A^e to B comes from proofs of strengthened thickness lemmas in [23, lemma 2.6] and [14, theorem X.3.9]. ⌟

Corollary 4.19. *If $C \neq \{n\}^A$ for all $n < e$ then* $\liminf_{s<\Gamma} \hat{R}(e+1,s) < \Gamma$.

Finally, the theorem is proved by induction. Assuming all requirements below (N_e) are satisfied, then all $(P_{e'}), e' < e$ are satisfied. It follows that the differences $\{B^{[\alpha]} \setminus A^{[\alpha]} : \alpha \in Q\}$ with $Q = p[e]$ are all ITTM-finite, and hence their finite union D is ITTM-finite. (It is decidable since one machine can simulate for each difference a machine deciding it, as there are only finitely many of these machines required. It is bounded as Γ is admissible [8, corollary 8.2] and hence closed under addition.) Therefore $A^e = \cup\{A^{[\alpha]} : \alpha \in Q\}$ reduces to $B^e = \cup\{B^{[\alpha]} : \alpha \in Q\}$. (Accept x if and only if $x \in B^e$ and $x \notin D$.)

Note that $\hat{I}_e$ reduces to A^e, since (N_e) is only injured by requirements $(P_{e'})$ with $e > e' \in p[Q]$. (The requirements of higher priority outside $p[Q]$ never act.) So given x, one can check $x \in \hat{I}_e$ by checking $x \in A^e$, and if so, running the construction until x is added to A and seeing whether (N_e) is thereby injured.

Hence $\hat{I}_e$ reduces to $\cup\{B^{[\alpha]} : \alpha \in Q\}$, and by assumption on C then C does not reduce to $\hat{I}_e$. The injury lemma yields the satisfaction of (N_e).

Assuming on the other hand that all requirements below (P_e) are satisfied, then all $(N_{e'}), e' \leq e$ are satisfied, and the window lemma ensures that $\hat{R}_e = \lim_{t<T} \hat{R}(e+1,t) < \Gamma$ exists, from which it follows that (P_e) is satisfied. Specifically, if $e = p(\alpha)$ we conclude

- $B^{[\alpha]} \setminus A^{[\alpha]} \subseteq \hat{R}_e$ so the difference D is bounded, and

- by the proof of the window lemma, there is a stage $t \in \mathsf{Won}$ so that $\hat{R}(e+1, t') = \hat{R}_e$ for all $t' \geq t$. Hence $A^{[\alpha]} \cap \hat{R}_e$ is decidable. It follows that $D = (B^{[\alpha]} \cap \hat{R}_e) \setminus (A^{[\alpha]} \cap \hat{R}_e)$ is decidable in B, and so the set of first appearances $F = \{s : (\exists d \in D)\, d \in B_s \wedge d \notin B_{<s}\}$ is bounded in Won: if it were unbounded, its supremum would be Γ and h could be reduced to B, since D is bounded and decidable in B. From the decidability and boundedness of F, it follows that D is decidable. □

By lemma 3.32, we find

Corollary 4.20. *Let $C \subseteq C' \subseteq \mathsf{Wrr}$ be semi-decidable and C decidable, and let $B \subseteq \mathsf{Won}$ semi-decidable with $B <_{\mathrm{ITTM}} h$. If C does not reduce to any finite number of columns of B, then there is a thick subset $A \subseteq B$ that C does not reduce to.*

Remark 4.21. The issue in translating the thickness lemma proof to ITTMs can also be seen in the light of true stages: traditionally, one would show that halting computations $\{e\}^A(w)$ are reflected in the true stages: there is always a true stage t with $\{e\}_t^{A_t}(w)$. Using techniques as the one in lemma 2.9, it can only be shown that there is some stage s with $\{e\}_t^{A_t}(w)$, even cofinally many, but not that they are true. For a computation to be reflected in a true stage t, all positively queried elements of A would have to be in A_t. ⌟

4.4 For p-OTMs

In the case of p-OTMs, the following can be said. We will only sketch the proof.

Theorem 4.22. *If B, C are p-OTM-semi-decidable and C does not p-OTM-reduce to $B^{[<\varepsilon]}$ for any ordinal ε, then there is a p-OTM-semi-decidable subclass A of B that C does not p-OTM-reduce to, and such that all $B^{[<\varepsilon]} \setminus A^{[<\varepsilon]}$ are p-OTM-finite. (In particular, all $B^{[\varepsilon]} \setminus A^{[\varepsilon]}$ are p-OTM-finite.)*

Proof sketch. The requirements are

$$\{\varepsilon\}^A \neq C \qquad (N_\varepsilon)$$

$$B^{[\varepsilon]} \setminus A^{[\varepsilon]} \text{ is p-OTM-finite} \qquad (P_\varepsilon)$$

prioritized alternatingly $(N_0) < (P_0) < (N_1) < (P_1) < \dots$ as before. Again define

$$u(\varepsilon, x, s) = \begin{cases} \sup\{y : y \text{ is queried in } \{\varepsilon\}_s^{A_s}(x)\} & \text{if } \{\varepsilon\}_s^{A_s}(x) \text{ halts;} \\ 0 & \text{otherwise.} \end{cases}$$

$$l(\varepsilon, s) = \sup\{x < s : (\forall y < x)\, C_s(y) = \{\varepsilon\}_s^{A_s}(y)\};$$

$$r(\varepsilon, s) = \sup\{u(\varepsilon, x, s) + 1 : x \leq l(\varepsilon, s)\};$$

and

$$a_s = \begin{cases} \min(A_s \setminus A_{<s}) & \text{if } A_s \setminus A_{<s} \neq 0; \\ \sup A_s \cup s & \text{otherwise.} \end{cases}$$

$$\{\hat{\varepsilon}\}_s^{A_s}(x) = \begin{cases} \{\varepsilon\}_s^{A_s}(x) & \text{if } u(\varepsilon, x, s) < a_s; \\ \uparrow & \text{otherwise.} \end{cases}$$

$$\hat{u}(\varepsilon, x, s) = u(\hat{\varepsilon}, x, s); \qquad \hat{l}(\varepsilon, s) = l(\hat{\varepsilon}, s); \qquad \hat{r}(\varepsilon, s) = r(\hat{\varepsilon}, s)$$

with minor differences to account for transfinitely many stages. A is defined by

$$\begin{aligned} A_0 &= \varnothing; \\ A_{s+1} &= A_s \cup \bigcup_{\varepsilon < s} B_{s+1}^{[\varepsilon]} \setminus \hat{R}(\varepsilon + 1, s); \\ A_\lambda &= \cup_{s<\lambda} A_s \end{aligned} \tag{4.4}$$

with $\hat{R}(\varepsilon, s) = \sup_{d<\varepsilon} \hat{r}(d, s)$. Essentially, nothing happens in limit stages. That each $A_s, s \in \mathsf{On}$ is decidable by an OTM in the parameter s (in particular, that A is thus OTM-semi-decidable) is an exercise left to the reader (cf. section 1.5).

Define the injury sets $\hat{I}_\varepsilon$ and its approximations

$$\begin{aligned} \hat{I}_\varepsilon &= \{x : (\exists s)\, x < \hat{r}(\varepsilon, s) \wedge x \in A_{s+1} \setminus A_s\} \\ \hat{I}_{\varepsilon,s} &= \{x : (\exists t \leq s)\, x < \hat{r}(\varepsilon, t) \wedge x \in A_{t+1} \setminus A_t\}. \end{aligned} \tag{4.5}$$

and the true stages T, seen cofinal in the ordinals as before.

The injury and window lemmas ($C \not\leq_{\text{p-O}} \hat{I}_\varepsilon \to C \neq \{\varepsilon\}^A$ and $C \neq \{\varepsilon\}^A \to (\exists \gamma)\ \lim_{\tau \in T} \hat{r}(\varepsilon, \tau) = \gamma$) are proved as for Turing machines, the only difference being in the way one attains reflection—but this is trivial as the class of queries made in a halting or bounded p-OTM-computations is a set (in L).

The rest of the proof is also similar, the only difference being that one has to deduce from $B^{[<\varepsilon]} \setminus A^{[<\varepsilon]} \subseteq \liminf \hat{R}(\varepsilon + 1, s)$ and the fact that A, B are p-OTM-semi-decidable that $B^{[<\varepsilon]} \setminus A^{[<\varepsilon]}$ is bounded and p-OTM-decidable. (Which follows from the observation that the *set* of first appearances cannot be cofinal in the stages.) □

We are not aware of any thickness lemma proved in α-recursion theory. As the queries made in a halting p-α-oracle-machine computation are never cofinal in α, it might be worth investigating whether a thickness lemma can be proved for for p-Γ-machines by adapting the proof of theorem 4.15.

Conclusion

Soare [23, section 3] has proved a variety of corollaries of the (strengthened) thickness lemma. Though we have not looked into this in detail, we believe that it is worth investigating whether our proof for a thickness lemma for certain sets of low degree can be modified as in [23, p. 520] to obtain a stronger thickness lemma, and whether a density theorem for these sets follows. We are hopeful in this regard, as in [23, p. 522], to find a A between $D <_{\mathrm{TM}} C$, the thickness lemma is applied only to $B \leqslant_{\mathrm{TM}} C$. But it is to be seen whether the rest of the proof can be generalized to ITTMs. The same question can be considered for p-OTMs.

For the splitting theorem, questions remain about the frequency of Clk-simple and Won-simple sets in the ITTM-semi-decidable degrees.

Finally, in light of our results, we would like to raise the following:

Question 2. *For which ITTM-semi-decidable $A \subseteq \mathsf{Won}$ are there $A' \subseteq B \subseteq \mathsf{Won}$ so that $A \equiv_{\mathrm{ITTM}} A'$ and B is ITTM-decidable? And for which $A \subseteq \mathsf{Wrr}$?*

Bibliography

[1] M. Carl, *Ordinal computability: an introduction to infinitary machines* (De Gruyter series in logic and its applications 9). De Gruyter, 2019, ISBN: 978-3-11-049562-1.

[2] J. C. E. Dekker, "A theorem on hypersimple sets," *Proceedings of the American Mathematical Society*, vol. 5, no. 5, pp. 791–796, 1954. DOI: 10.2307/2031868.

[3] G. C. Driscoll, "Metarecursively enumerable sets and their metadegrees," *The Journal of Symbolic Logic*, vol. 33, no. 3, pp. 389–411, 1968. DOI: 10.2307/2270325.

[4] L. Galeotti, E. S. Lewis, and B. Löwe, "Symmetry for transfinite computability," in *Unity of Logic and Computation*, G. Della Vedova, B. Dundua, S. Lempp, and F. Manea, Eds., Springer Nature Switzerland, 2023, pp. 65–76. DOI: 10.1007/978-3-031-36978-0_6.

[5] J. D. Hamkins and A. Lewis, "Post's problem for supertasks has both positive and negative solutions," *Archive for Mathematical Logic*, vol. 41, no. 6, pp. 507–523, 2002. DOI: 10.1007/s001530100112.

[6] J. D. Hamkins and A. Lewis, "Infinite time Turing machines," *The Journal of Symbolic Logic*, vol. 65, no. 2, pp. 567–604, 2000. DOI: 10.2307/2586556.

[7] J. D. Hamkins and R. G. Miller, "Post's problem for ordinal register machines: An explicit approach," *Annals of Pure and Applied Logic*, Computation and Logic in the Real World: CiE 2007, vol. 160, no. 3, pp. 302–309, 2009. DOI: 10.1016/j.apal.2009.01.004.

[8] J. D. Hamkins and D. E. Seabold, "Infinite time Turing machines with only one tape," *Mathematical Logic Quarterly*, vol. 47, no. 2, pp. 271–287, 2001. DOI: 10.1002/1521-3870(200105)47:2<271::AID-MALQ271>3.0.CO;2-6.

[9] T. J. Jech, *Set theory* (Springer monographs in mathematics), The 3rd millennium ed., rev. and expanded. Springer, 2003, ISBN: 978-3-540-44085-7.

[10] P. Koepke, "Turing computations on ordinals," *Bulletin of Symbolic Logic*, vol. 11, no. 3, pp. 377–397, 2005. DOI: 10.2178/bsl/1122038993.

[11] P. Koepke, "Ordinal computability," in *Mathematical Theory and Computational Practice*, K. Ambos-Spies, B. Löwe, and W. Merkle, Eds., ser. Computation and Logic in the Real World: CiE 2009, vol. 5635, Springer Berlin Heidelberg, 2009, pp. 280–289. DOI: 10.1007/978-3-642-03073-4_29.

[12] P. Koepke and B. Seyfferth, "Ordinal machines and admissible recursion theory," *Annals of Pure and Applied Logic*, Computation and Logic in the Real World: CiE 2007, vol. 160, no. 3, pp. 310–318, 2009. DOI: 10.1016/j.apal.2009.01.005.

[13] P. Odifreddi, *Classical Recursion Theory* (Studies in Logic and the Foundations of Mathematics). North-Holland, 1992, vol. 125, ISBN: 978-0-444-87295-1.

[14] P. Odifreddi, *Classical Recursion Theory, Volume II* (Studies in Logic and the Foundations of Mathematics). Elsevier, 1999, vol. 143, ISBN: 978-0-444-50205-6.

[15] E. L. Post, "Recursively enumerable sets of positive integers and their decision problems," *Bulletin of the American Mathematical Society*, vol. 50, no. 5, pp. 284–316, 1944. DOI: 10.1090/S0002-9904-1944-08111-1.

[16] G. E. Sacks and S. G. Simpson, "The α-finite injury method," *Annals of Mathematical Logic*, vol. 4, no. 4, pp. 343–367, 1972. DOI: 10.1016/0003-4843(72)90004-6.

[17] G. E. Sacks, "On the degrees less than $0'$," *Annals of Mathematics*, vol. 77, no. 2, pp. 211–231, 1963. DOI: 10.2307/1970214.

[18] G. E. Sacks, *Higher Recursion Theory* (Perspectives in Logic). Cambridge University Press, 2017, vol. 2. DOI: 10.1017/9781316717301.

[19] J. R. Shoenfield, *Degrees of unsolvability* (North-Holland mathematics studies 2). North-Holland Publ. Co. etc., 1971, ISBN: 978-0-7204-2061-6.

[20] R. A. Shore, "Splitting an α-recursively enumerable set," *Transactions of the American Mathematical Society*, vol. 204, pp. 65–77, 1975. DOI: 10.2307/1997349.

[21] R. A. Shore, "Splitting an α-recursively enumerable set," *Transactions of the American Mathematical Society*, vol. 204, pp. 65–77, 1975. DOI: 10.2307/1997349.

[22] R. A. Shore, "The recursively enumerable α-degrees are dense," *Annals of Mathematical Logic*, vol. 9, no. 1, pp. 123–155, 1976. DOI: 10.1016/0003-4843(76)90007-3.

[23] R. I. Soare, "The infinite injury priority method," *The Journal of Symbolic Logic*, vol. 41, no. 2, pp. 513–530, 1976. DOI: 10.2307/2272252.

[24] S. A. Terwijn, *Syllabus Computability Theory*. 2022. eprint: `https://www.math.ru.nl/~terwijn/teaching/syllabus.pdf`.

[25] A. M. Turing, “On computable numbers, with an application to the Entscheidungsproblem,” *Proceedings of the London Mathematical Society*, 2nd ser., vol. 42, no. 1, pp. 230–265, 1937. DOI: `10.1112/plms/s2-42.1.230`.

[26] P. D. Welch, “The length of infinite time Turing machine computations,” *Bulletin of the London Mathematical Society*, vol. 32, no. 2, pp. 129–136, 2000. DOI: `https://doi.org/10.1112/S0024609399006657`.

[27] J. Winter, “Space complexity in infinite time Turing machines,” M.Sc. thesis, Universiteit van Amsterdam, 2007.

Appendix A

ITTM-computability

Sections A.1 and A.3 contain proofs of technical results used in the thesis, while A.2 provides background on the decidability of the writable well-orders.

A.1 In-order generation of Won

Recall that codes for ordinals are well-orders on a subset of ω.

Lemma A.1. *There is an ITTM which, given a code for α, generates a code all predecessors of α in ordinal order.*

Proof. This is based on [6, theorem 2.2]. Let x be a code for an ordinal α, and let R be the relation it represents. The ITTM works as follows: it has an extra "strike-off tape" initially filled with 0s. Call n stricken off if the nth cell of the strike-off tape is 1. The ITTM first determines the domain of R (by checking for each $n < \omega$ whether there is some $m < \omega$ so that the $\langle n, m\rangle$th bit of x is 1, for example), and strikes off all n outside of it. As long as the strike-off tape contains 0s, the ITTM searches for the least n under R that is not stricken off, creates a code y for the order type of the predecessors of n under R by taking x and removing all mention of m above n, and shows it on the output tape. (That is, it first writes $0y$ on the output tape, and then flashes the first cell to write $1y$, following the definition of ITTM-generability.) It then strikes off n. When the ITTM halts, it has written a code for every $\beta < \alpha$ in ordinal order. □

Note that in the proof above, no mention is made of compound limit steps. Formally, the ITTM has to detect these steps (as in example 1.6) and clean up any tapes used for intermediate results (used while finding the next minimal element, for example). It is good to note that the strike-off tape is not affected by limit behavior, as cells only change value once, and that the first cell of the output tape is always 0 on (compound) limit steps, so no scrambled results are accidentally generated.

Corollary A.2. *There is an ITTM which generates* Won *in ordinal order.*

Proof. As Wrr is ITTM-generable, WO is ITTM-decidable, and canonical codes can be computed from codes for writable ordinals, Won is generable. We have to show that there is an ITTM that generates it in ordinal order.

Again consider an ITTM with an extra strike-off tape. It simultaneously simulates all computations $\{e\}(0)$, as in e.g. [1, theorem 2.5.15]. At the start of every simulation step, the machine checks which machines E have just halted. If so, the ITTM searches for a code for the halting time α of one such $\{e\}(0)$ (the least $e \in E$, say). This it can do by generating Won until a code x for an ordinal $\beta \geq \alpha$ is found (which an ITTM can check by, for each β generated, simulating $\{e\}(0)$ while counting through β, so that one simulation step is executed for every minimal element removed from the well-order coded by x). Such a β must be found as all clockable ordinals are writable, and once found, the ITTM can by lemma A.1 (and repeating the process above) find the least such β, which is α.

Once a code x for α is found, it simultaneously simulates all computations $\{m\}(0)$ for all m stricken. For each simulation step, it removes a minimal element of the relation R coded by x. (By another striking procedure, say.) Once all $\{m\}(0)$ have halted, the natural numbers left in R are precisely those whose sets of predecessors originally had an order type greater than α. For each of those, a canonical code is shown on the output tape (which can be done in ordinal order by lemma A.1), and then also a code for α itself. Finally, all elements of E are stricken off and the machine continues simulating the computations for machines that have not been stricken off.

As cells on the strike-off tape change value at most once, its content is not scrambled at (compound) limit steps. By design of the ITTM, all ordinals generated are generated in ordinal order. As the clockable ordinals are cofinal in the writable ordinals (and Γ is a limit ordinal), this process eventually generates the writable ordinals in canonical order. □

A.2 The undecidability of the writable well-orders

Proposition A.3. *The set W of writable reals coding ordinals is not ITTM-decidable.*

Proof. We will assume that W is ITTM-decidable and derive the contradiction that $\Gamma = \sup \mathsf{Won} = \sup \mathsf{Clk}$ is writable. If α is an ordinal, define

$$h_\alpha = \{m : \{m\}_\alpha(0)\downarrow \wedge (\forall n < m)(\exists \beta \leq \alpha)\, \neg(\{m\}_\beta(0)\downarrow \leftrightarrow \{n\}_\beta(n)\downarrow)\}$$

for the set of least ITTMs representing $\mathsf{Clk} \cap \alpha$. Γ is the first ordinal so that h_Γ is not ITTM-computable. Given α, define the following well-order $\prec_\alpha$ on ω:

- if $m, n \notin h_\alpha$ then $m \prec_\alpha n$ iff $m < n$
- if $m \notin h_\alpha$ and $n \in h_\alpha$, then $m \prec_\alpha n$
- if $m, n \in h_\alpha$ then $m \prec_\alpha n$ if $\{n\}(0)$ halts before $\{n\}(0)$.

If α is writable, then a code for $(\omega, \prec_\alpha)$ is writable, and so its order type is a writable ordinal (and by the below, often not a clockable ordinal). The order type of $\prec_\Gamma$ is that of the clockable ordinals, and since this is the same as the order type of the writable ordinals [6], the order type of $\prec_\Gamma$ is Γ.

Consider the ITTM which simultaneously simulates all $\{m\}(0)$ and uses this to create codes for $\prec_\alpha$ for increasing $\alpha < \Gamma$, continuously refining an approximation for $\prec_\Gamma$. For each approximation $\prec$, which is seen to always be some $\prec_\alpha$ with $\alpha \leq \Gamma$, it checks whether $(\omega, \prec_\alpha)$ is a writable ordinal, which it can do by the assumption that W is ITTM-decidable. It writes the first approximation that is not in W and then halts, writing a code for Γ. □

Further inspection of the proof above gives the following corollaries.

Corollary A.4. *The set of reals coding writable ordinals is not ITTM-decidable.*

Corollary A.5. *The weak halting problem h reduces to W, and hence, since W is semi-decidable, $h \equiv_{\mathrm{ITTM}} W$.*

Since the set WO of reals coding well-orders is ITTM-decidable and W is the intersection of Wrr and WO, we also obtain

Corollary A.6. $h \equiv_{\mathrm{ITTM}} \mathsf{Wrr}$. *In particular,* Wrr *is undecidable.*

A.3 Ordinal arithmetic on (non-writable) codes

Lemma A.7. *There is an ITTM that, given codes x, y for (not necessarily writable) ordinals α, β, computes a code for $\langle \alpha, \beta \rangle$.*

Proof. By clocking both ordinal simultaneously, an ITTM can determine if $\alpha \geq \beta$, and thus it can determine a code z for $\gamma = \max\{\alpha, \beta\}$ (it is either x or y).

z codes a well-order $\preceq_z$ on a subset D_z of ω with order type γ, so that $n \preceq_z m$ if and only if the $\langle n, m \rangle$th bit of z is a 1.

The ITTM then makes a code w for $\gamma + 1$ as follows: the $\langle n+1, m+1 \rangle$th bit of w is equal to the $\langle m, n \rangle$th bit of z, and for all $n < \omega$ the $\langle n, 0 \rangle$th bit of w is 1. (So, in the order induced by w, 0 is now the new maximal element.)

Write D for the domain of the well-order $\preceq$ induced by w, define $\operatorname{pred} u = \{n \in D : n \prec un\}$ (where $\prec$ is the strict version of $\preceq$) and let $a, b \in D$ be such that $(\operatorname{pred} a, \preceq)$ has order type α, and $(\operatorname{pred} b, \preceq)$ has order type β. The ITTM determines a, b by iteration over D (all $n < \omega$ so that the $\langle n, n \rangle$th bit of w is 1) and clocking.

The ITTM now iterates over all $d, e, f, g < D$, and writes a 1 on the $\langle\langle d, e \rangle, \langle f, g \rangle\rangle$th cell of the output tape if one of

- $\max_\preceq \{d, e\} \prec \max_\preceq \{f, g\}$, or
- $\max_\preceq \{d, e\} = \max_\preceq \{f, g\} \wedge d \prec f$, or

- $\max_{\preceq}\{d,e\} = \max_{\preceq}\{f,g\} \wedge d = f \wedge e \prec g$

hold, and furthermore (intuitively, to exclude pairs greater than $\langle\alpha,\beta\rangle$)

- $\max_{\preceq}\{d,e\} \prec \max_{\preceq}\{a,b\}$, or
- $\max_{\preceq}\{d,e\} = \max_{\preceq}\{a,b\} \wedge d \prec a$, or
- $\max_{\preceq}\{d,e\} = \max_{\preceq}\{a,b\} \wedge d = a \wedge e \prec b$

holds, and if also one of the above holds, where d, e is replaced with f, g. The ITTM can check the above simply because $s \preceq t$ holds if and only if the $\langle s,t\rangle$th bit of w is 1: for each quadruple d, e, f, g the above procedure is even TM-computable, the whole process can be done in ω steps.

Comparing the conditions above with the definition of the Cantor pairing function, we find that the real v on the output tape codes a well-order $\preceq_z$ on ω, the domain of which consists of the pairs $\langle d,e\rangle$ so that, identifying sets with their order types, $\langle(\operatorname{pred} d,\preceq),(\operatorname{pred} e,\preceq)\rangle \leq \langle\alpha,\beta\rangle$, and $\langle d,e\rangle \preceq_z \langle f,g\rangle$ if and only if $\langle(\operatorname{pred} d,\preceq),(\operatorname{pred} e,\preceq)\rangle \preceq_z \langle(\operatorname{pred} f,\preceq),(\operatorname{pred} g,\preceq)\rangle$. In other words: it has order type $\langle\alpha,\beta\rangle$. □

Appendix B

Machine models

This appendix sketches proofs on the equivalence of machine model definitions, and gives some insight in the relationship between ITTMs an p-Γ-machines.

B.1 Limit rules

Originally, Hamkins and Lewis defined the limit behavior of ITTMs as follows: the machine head is on the first cell, and each cell contains the $\limsup$ of its previous values. Below, we motivate that a partial function on the reals is computable by an ITTMs as defined in section 1.1 if and only if it is computable by an ITTM as defined originally.

Formal proofs show that a machine of one kind can emulate a machine of the other kind, and this either involves constructing a universal machine or describing computable modifications of machine codes. This can be rather terse. Below, proof sketches with the key ideas are presented.

Lemma B.1. *The* $\liminf$ *cell-value behavior can be emulated in a* $\limsup$ *world, and vice versa.*

Proof sketch. Let each cell c be represented by two cells c_0, c_1. This can be emulated by doubling the number of tapes, for example. If a machine would normally write a 1 on c, it writes 1 on c_0 and 0 on c_1. If it would normally write a 0 on c, it writes a 0 on c_0 and a 1 on c_1. In limit stages, there are three possibilities:

- $(c_0, c_1) = (1, 0)$, in which case c would contain a 1
- $(c_0, c_1) = (0, 1)$, in which case c would contain a 0
- Otherwise, the value of c would not have converged, and (c_0, c_1) would be $(1, 1)$ in a $\limsup$ world and $(0, 0)$ in a $\liminf$ world.

Hence, an ITTM living in a $\limsup$ world could simulate $\liminf$ behavior by, after limit steps, changing all $(1, 1)$-pairs into $(0, 1)$-pairs, and an ITTM living in a lim inf world could simulate lim sup behavior by, after limit steps, changing all $(0,0)$-pairs into $(1,0)$-pairs.

That this works for non-compound limit steps is apparent, but it also works with compound limits: if after every limit step a "clean up" occurs, $(0,0)$- or $(1,1)$-pairs are found in compound limit steps α if and only if the value of the cell they represent would have diverged cofinally often before α. □

So if runtime is not an issue, the two behaviors are equivalent. Similarly

Lemma B.2. *The head-to-start limit behavior can be emulated in a head-to-*lim inf *world, and vice versa.*

Proof sketch. To emulate head-to-lim inf in a head-to-start world with lim inf cell-value behavior, machines can be altered to have an extra "position tape", on which it keeps track of its head position: it starts by writing a 1 on the first cell of the position tape. Whenever it moves right, it writes a 1 on the new cell of the position tape it is on, and whenever it moves left, it writes a 0 on the position tape before moving left. Thus, informally speaking, for the first ω steps in the computation, the machine is at cell n if and only if only the first n cells of the position tape contain a 1. (This can be made formal by speaking in terms of an original machine and the modified machine, obtained by doing the extra work of keeping track of position.)

At limit steps then, the number of 1s on the position tape corresponds with the lim inf of the cell positions (because of the lim inf cell-value behavior). Hence the machine can informally resume from this head position by (starting from the head) moving to the end of the 1s on the position tape. (Formally, in the limit state of the modified machine, move to the end of the 1s of the position tape, and then enter the limit state of the original machine.) It follows that this leads to the correct behavior.

To emulate head-to-start behavior in a head-to-lim inf world, add an extra "start tape" which contains a 1 only in the first cell. On limit steps, first move to the start by inspecting said tape. □

We conclude:

Proposition B.3. *For every ITTM e as defined in section 1.1 there is an ITTM e' as defined in [6] that induce the same partial function on the reals, and vice versa. (Furthermore, this is "uniform in codes", in the sense that there is a Turing-computable function turning e into an e' for any given e, and vice versa.)*

There is also a discrepancy in the definition of α-machines and OTMs defined in section 1.1 and their original definitions: originally, there was no notion of a limit state, but the state s in limit steps α was given by the lim inf of the states taken up to α, and the position was given as the lim inf of the position the machine was in while in state s up to α. That our machines can simulate the original behavior and vice versa, can be seen by using similar tricks as above.

(With a “state tape” and “position-in-state-s tapes” in one direction, and by e.g. watching for limit steps in the other. Since admissible ordinals are closed under ordinal exponentiation, such manipulations do not lead to running out of time.)

B.2 Γ-machines, p-Γ-machines, and ITTMs

Recall that an ordinal is ITTM-writable if and only if it lies below the ordinal Γ. In [6, corollary 8.2], Hamkins and Lewis show that Γ is an admissible ordinal. In this section, we show some similarities between ITTMs and Γ-machines.

Lemma B.4. *There is an ITTM-computable partial function $u(x, y, z)$ so that: if x, y are codes for $\varepsilon, \alpha < \Gamma$, then there is a canonical code z for a writable ordinal τ with $u(x, y, z)\downarrow$ if and only if the p-Γ-computation $\{\varepsilon\}(\alpha)$ halts, and for all z that code writable ordinals for which $u(x, y, z)\downarrow$ it is the case that $u(x, y, z)$ is a code for the result of $\{\varepsilon\}(\alpha)$.*

Proof sketch. Recall that, by the results cited in section 1.5, an ITTM can generate L_Γ effectively, and that halting p-Γ-computations are elements of L_Γ. So there is an ITTM that “simply” searches for halting computations. □

Corollary B.5. *If a partial function $f : \Gamma \to \Gamma$ is p-Γ-computable, then there is a partial ITTM-computable $g : 2^\omega \to 2^\omega$ so that, up to coding, $g \cap \mathsf{Won}^2 = f$.*

A converse also holds, even for parameter-free Γ-machines.

Lemma B.6. *If a partial function $g : 2^\omega \to 2^\omega$ is ITTM-computable, then there is a Γ-computable partial function $f : \Gamma \to \Gamma$ so that, up to coding, $f = g \cap \mathsf{Won}^2$.*

Proof sketch. Recall that ITTM-computations on writable input halt if and only if they halt in clockable time, so they halt if and only if they halt in less than Γ steps. So, on writable inputs, a halting ITTM-computation can be carried out by a 3-tape Γ-machine by simply following the instructions of the ITTM, with the one modification that, if the Γ-machine ever finds itself on the ωth cell, it has to move left once to be at the start of the tape before continuing the computation. (To recognize that it is on the ωth cell in a limit step, it can, for example take an extra tape with a 1 only on its first cell.) □

Corollary B.7. *A partial function $f : \Gamma \to \Gamma$ is p-Γ-computable if and only if it is Γ-computable.*

Corollary B.8. *A partial function $f : \Gamma \to \Gamma$ is Γ-computable if and only if there is a partial ITTM-computable $g : 2^\omega \to 2^\omega$ so that, up to coding, $g \cap \mathsf{Won}^2 = f$.*