Structural Properties of the Local Turing Degrees

James Riley

Submitted in accordance with the requirements

for the degree of Doctor of Philosophy

The University of Leeds

Department of Pure Mathematics

April 2017

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Joint Work

Chapter 4 contains work from the paper *Computably Enumerable Turing Degrees and the Meet Property* jointly authored with Benedict Durrant, Andrew Lewis-Pye, and Keng Meng Ng. The main result in this paper was primarily my work with some assistance from Benedict Durrant. It was based on an idea by Ng and Lewis-Pye, and important technical advice was given by Lewis-Pye.

Acknowledgements

I would like to thank my supervisors Barry Cooper and Andrew Lewis-Pye for their support and guidance during my PhD. I would also like to thank my parents and my partner for getting me through the difficult times.

Abstract

In this thesis we look at some properties of the local Turing Degrees, as a partial order. We first give discussion of the Turing Degrees and certain historical results, some translated into a form resembling the constructions we look at later.

Chapter 1 gives a introduction to the Turing Degrees, Chapter 2 introduces the Local Degrees. In Chapter 3 we look at minimal Turing Degrees, modifying some historical results to use a priority tree, which we use in chapter 4 to prove the new result that every c.e. degree has the (minimal) meet property. Chapter 5 uses similar methods to establish existence of a high_2 degree that does not have the meet property.

Contents

Chapter 1

Introduction

Turing [Tur36] created the truly convincing model of what it means for an algorithm to be computable, by introducing the Turing Machine. Other sufficiently strong notions of computability, such as the general recursive functions or Church's lambda calculus have been shown to be equivalent. Notions that are strictly stronger, such as allowing ω calculations in finite time, would seem to be stronger than what a human can accomplish. So for now we are comfortable accepting Turing computability as the definition of computability.

Given a Turing machine we say that it decides or computes a set A of natural numbers if given input x it outputs a 1 if $x \in A$ and a 0 if $x \notin A$. A set A is computable if there is some Turing machine that computes it. Turing's great discovery was that very few sets are actually computable. Only countably many sets are computable, while there are uncountably many sets of natural numbers.

The construction of the Turing Machine leads naturally to a Gödel numbering on all Turing Machines from which we order all Turing Machines $\{\Psi_e : e \in \mathbb{N}\}$, by ascending Gödel number.

Given that there are vastly more incomputable sets than computable sets, we

seek to put some structure to them, and then to study that structure. Turing [Tur39] introduced the Oracle Turing Machine, which as well as the standard instructions can query an oracle about membership of one set. We then say that a set A is computable from B if A is computable by some Oracle Turing Machine with oracle B. Post [Pos03] used this to define the structure of the Turing degrees by defining a partial ordering $\leq_T$ where $A \leq_T B$ iff A is computable from B. If $A \leq_T B$ and $B \leq_T A$ then we say A and B are *Turing equivalent* and write $A \equiv_T B$. This is an equivalence relation and we denote the equivalence class containing A by $\mathbf{a}$. If there is $A \in \mathbf{a}$ and $B \in \mathbf{b}$ such that $A \leq_T B$ then we say $\mathbf{a} \leq \mathbf{b}$ in the structure $\mathcal{D}$. Studying the structure of $\mathcal{D}$ is a project that has continued since Kleene and Post introduced it. Clearly there is a least element consisting of exactly the computable degrees, which as the least element we denote $\mathbf{0}$.

Given our listing of Turing functionals $\{\Psi_e : e \in \mathbb{N}\}$ it is natural to ask if the computation $\Psi_e(x)$ halts or not. The set $\{(e, x) : \Psi_e(x) \downarrow\}$ is a perfectly valid set of pairs of natural numbers to consider, and it turns out to be Turing equivalent to $\{e : \Psi_e(e) \downarrow\}$. This problem is known as the *halting problem* since each question is whether a given Turing machine halts on a given input. The degree containing the halting problem is known as $\mathbf{0}'$. The degrees that are below $\mathbf{0}'$ are known as the *local degrees* and have certain properties that make them easier to study than a general degree. For instance, every local degree, A, has a Δ_2-approximating sequence, which is a computable function $A(s, x)$ with the property that $lim_{s\to\infty} A(s, x) = A(x)$.

Some familiarity with the basic notions discussed so far will be assumed. Precise definitions can be found in [Coo03] or [Odi92],[Odi99].

1.1 Notation and Conventions

While Computability Theory is a relatively young discipline in mathematics some notations have not yet been standardised. We therefore set out the notations we shall use in this thesis.

We use lower case Roman letters from the end of the alphabet (n, x, y, z) to denote natural numbers, and capital Roman letters to denote subsets of $\mathbb{N}$. We use lower case Greek letters $(\lambda, \gamma, \nu, \mu, \sigma, \tau)$ to denote finite strings, which will be typically be members of $2^{<\omega}$ but instead may be strings in $\omega^{<\omega}$. We let λ denote the unique empty string, and given two strings σ, τ we let $\sigma * \tau$ be their concatenation. Given a string σ we write $|\sigma|$ for the length of σ. We then write $\sigma \upharpoonright n$ for the initial segment of σ of length n, as long as $n \leq |\sigma|$. We write σ^- for $\sigma \upharpoonright |\sigma| - 1$, if σ is not λ. We write $\sigma^\dagger$ for the string identical to σ, except for the final bit. We write $\sigma \subset \tau$ if τ extends σ. If $\sigma \not\subset \tau$ and $\tau \not\subset \sigma$ then σ is incompatible with τ, written $\sigma|\tau$. We write $\sigma \wedge \tau$ for the greatest lower bound of σ, τ, i.e. the longest initial segment of σ and τ that is identical. This may be the empty string.

Possibly partial functions $\mathbb{N} \to \mathbb{N}$ shall have lower case Roman letters starting from f. If f is defined on input n then we write $f(n)\downarrow$, otherwise we write $f(n)\uparrow$. At times we will be considering functions as Turing functionals, and we use capital Greek letters for Turing functionals. In particular we fix some standard listing of the Turing functionals and write this list $\{\Psi_e : e \in \mathbb{N}\}$. To make some arguments easier, we will insist on Ψ_0 being the identity functional. If the functional Ψ with oracle A and input x halts in at most s stages and outputs y then we write $\Psi(A; x)[s]\downarrow = y$. Extending this notation we write $\Psi(A)$ for the possibly partial function which on argument x is equal to $\Psi(A; x)$. We identify subsets of $\mathbb{N}$ with their characteristic functions, so we may say $\Psi(A) = B$. We extend the notation to strings and say

$\Psi(\sigma; x)[s] \downarrow$ if the oracle computation converges where σ is used as an oracle, and any oracle query greater than $|\sigma|$ makes the computation diverge. A function is total if for all x $f(x) \downarrow$. If a function is not total then it is partial. The *use* of an oracle computation $\Psi(A; x)$ is $n + 1$ where n is the largest number such that n is queried in the oracle computation. We write $u(A, e, x)$ for the use in computing $\Psi_e(A; x)$. Given a finite string σ then we may write $\Psi(\sigma; x)[s] \downarrow = y$ if Ψ outputs y after s stages with σ acting as an oracle, with use less than $|\sigma|$. If $\Psi_e(A, x) \downarrow$ for $x < l$ then we write $\Psi_e(A) \upharpoonright l \downarrow$.

Calligraphic letters ($\mathcal{P}, \mathcal{Q}$) shall denote requirements within a priority argument, or the node on a tree of strategies that is trying to satisfy that requirement. The exception is $\mathcal{D}$, which is the partial ordering of the Turing Degrees.

We fix some computable bijection $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and write it $\langle x, y \rangle = n$. It has the property that we can nest it if we require a bijection $\mathbb{N}^n \to \mathbb{N}$ by nesting the bijection $\langle x_0, \cdots, \langle x_{n-2} \langle x_{n-1}, x_n \rangle \rangle \cdots \rangle$. We use the convention that $\langle a, b \rangle < \langle a, b + 1 \rangle$.

A set A is computably enumerable (c.e.) if there is an effective process for enumerating all members of A. Equivalently, there is some computable function f such that $A =$range(f). The important point for c.e. sets is that once enumerated into the set an element may never be removed from the set.

The Latin letter i will denote an element in $\{0, 1\}$, and $\bar{i}$ will denote $1 - i$.

For all $n \in \mathbb{N}$ we denote the n-th prime by p_n.

1.2 Trees

A binary tree is a possibly partial function $T : 2^{<\omega} \to 2^{<\omega}$, subject to the following conditions; for $\sigma \in 2^{<\omega}$ and $i \in \{0, 1\}$. If $T(\sigma * i) \downarrow$ then:

1. $T(\sigma) \downarrow \subset T(\sigma * i)$;

2. $T(\sigma * \bar{i}) \downarrow |T(\sigma * i)$.

If T is a total function then we describe the tree as perfect.

We identify T with its range. So we may say τ lies on T, meaning that τ is in the range of T. We may write this $\tau \in T$. Given a set A we say A is a branch of T, or A lies on T, if for infinitely many $\alpha \subset A$, $\alpha \in T$.

A tree T_0 is a subtree of tree T_1 if the range of T_0 is a subset of the range of T_1, and we may write this $T_0 \subseteq T_1$. τ is of level n, or of height n, in a tree T if $\tau = T(\sigma)$ where $|\sigma| = n$. The string τ is a leaf of T if $\tau \in T$ and no extension of τ is in T. A tree is of height n if it has finite range and all leaves are of height n.

A tree is partial computable (p.c.) if T is partial computable. This means that the range of T is c.e.. Given a p.c. tree T we let $T[s]$ be the portion of the tree enumerated by stage s. We shall require that $T[s]$ is a tree for all s.

Within a tree T we call τ a successor of σ if $\sigma, \tau \in T$ and $\sigma \subset \tau$. We also say that σ is a predecessor of τ. If there is no string $\gamma \in T$ such that $\sigma \subset \gamma \subset \tau$ then we say τ is the immediate successor of σ and σ is the immediate predecessor of τ. If σ is not the empty string then we use σ^- to denote the immediate predecessor of σ.

1.3 Priority Arguments

Discovering anything about the Turing degrees usually requires building a set within a Turing degree that demonstrates that a property does or does not hold. For example in the Friedberg-Muchnik proof [Fri57],[Muc56] that there exist incomputable c.e. degrees we create two c.e. sets A,B which are Turing incomputable. We do not have an easy way of ensuring that two degrees be incompatible, so we break it down into manageable chunks. Logically, saying that A and B are incompatible is the same as saying that there is no functional Ψ such that $\Psi(A) = B$ or $\Psi(B) = A$.

Which is then to say that for all e, $\Psi_e(A) \neq B$ and $\Psi_e(B) \neq A$. This only requires one x for which $\Psi_e(A;x) \neq B(x)$ and one x for which $\Psi_e(B;x) \neq A(x)$, where we consider the case $\Psi_e(A;x) \uparrow \neq B(x)$ for any value of $B(x)$. We've now turned two unwieldy requirements into two countable lists of manageable requirements. For clarity, let's label these requirements:

$$\mathcal{P}_e : (\exists x)\Psi_e(A;x) \neq B(x)$$

$$\mathcal{R}_e : (\exists x)\Psi_e(B;x) \neq A(x)$$

Then as long as we satisfy $\{\mathcal{P}_e : e \in \mathbb{N}\}$ and $\{\mathcal{R}_e : e \in \mathbb{N}\}$ we will satisfy the theorem. We usually now look at how we can satisfy one requirement in isolation.

To satisfy an individual requirement $\mathcal{P}_e$ we fix a value x on A,B that we are looking at, and set $A(x) = B(x) = 0$. If in some stage s we observe $\Psi_e(A;x)[s] \downarrow = 0$ then we set $B(x) = 1$ and satisfy the requirement. If we never observe this then we leave $B(x) = 0$ and the requirement is satisfied. Requirement $\mathcal{R}_e$ works the same way with the roles of A,B reversed.

The problem emerges when there is more than one requirement. If we have requirements $\mathcal{P}_0$ and $\mathcal{P}_1$ and both choose the same x, then we have the issue that one might want to change $B(x)$ and the other wants to leave it fixed. With requirements $\mathcal{P}_0$ and $\mathcal{R}_0$ one might observe that $\Psi_0(A;x) \downarrow = 0$ and change $B(x)$, while at a later stage $\mathcal{R}_0$ changes A so it is no longer the case that $\Psi_0(A;x) \downarrow = 0$.

The solution is to prioritise the requirements. Cooper [Coo03] compares this to a lunch queue where we place $\mathcal{P}_0$ ahead of $\mathcal{R}_0$ ahead of $\mathcal{P}_1$ in a queue for lunch. $\mathcal{R}_0$ can take its lunch if $\mathcal{P}_0$ has already taken its lunch (has set $B(x) = 1$), or if it accepts that at some later stage $\mathcal{R}_0$ might have to return to the lunch queue if $\mathcal{R}_0$ takes its lunch and later $\mathcal{P}_0$ takes his lunch. In this case we give the priority

ordering:

$$\mathcal{P}_0 <_L \mathcal{R}_0 <_L \mathcal{P}_1 <_L \cdots <_L \mathcal{P}_e <_L \mathcal{R}_e <_L \cdots$$

The problem as described is *injury* and through one method or another we ensure that no requirement is allowed to injure a requirement of higher priority. So no requirement is allowed to injure $\mathcal{P}_0$, if it chooses $B(x) = 1$ then nobody else is allowed to say otherwise, and similarly nobody else is allowed to change A up to the use of the computation $\mathcal{P}_0$ observes. $\mathcal{R}_0$ on the other hand, can only be injured by $\mathcal{P}_0$. In this argument each requirement can injure lower priority arguments once, and there are finitely many requirements of higher priority than any requirement. So this argument is *finite injury*. Other arguments may involve *infinite injury*.

Traditionally we would consider the requirements ordered in a straight line and enforce the priority ordering with a restraint function, which is essentially a function that $\mathcal{P}_e$ tells is its section of A, B and lower priorities must know this. In this work we are interested in the tree priority method introduced by Lachlan [Lac76] and developed by Harrington [Har82] so we will not give Friedberg and Muchnik's proofs, but instead a modified proof using the tree method. We do this to illustrate the tree method, and to give a template of how such proofs will be given.

1.4 Friedberg-Muchnik

Theorem 1. *There exist incompatible c.e. degrees.*

Proof. We build c.e. sets A, B, such that $A \not\leq_T B$ and $B \not\leq_T A$. We do this by satisfying the following requirements:

$$\mathcal{P}_e : (\exists x)\Psi_e(A; x) \neq B(x)$$

$$\mathcal{R}_e : (\exists x)\Psi_e(B; x) \neq A(x)$$

1.4.1 Construction

We set the requirements on a tree of strategies where the base node is $\mathcal{P}_0$ and every node has two outcomes $0 <_L 1$. Node $\mathcal{P}_e$ will have two successors which are both $\mathcal{R}_e$ and node $\mathcal{R}_e$ will have two successors that are both copies of $\mathcal{P}_{e+1}$

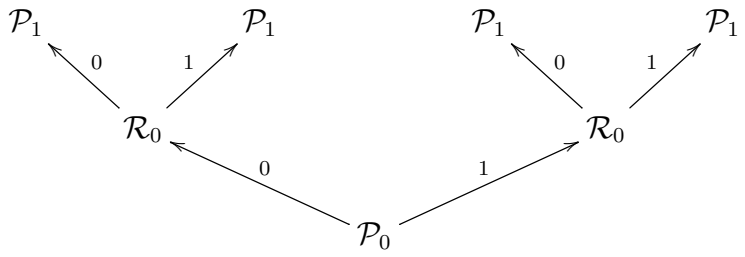

Figure 1.1: Part of the tree of strategies

In this construction, when we say to pick "x large" we mean to take x larger than any number yet considered in the construction.

We now define what a node does on its first action. A node $\mathcal{P}_e$ or $\mathcal{R}_e$ picks x large and sets $A(x) = B(x) = 0$. It then halts the stage.

At stage s where a node $\mathcal{P}_e$ is not acting for the first time, if it has ever played outcome 1 then it plays outcome 1. Otherwise, if it has never acted before, or has acted but never played outcome 1, it checks if

$$\Psi_e(A; x)[s] \downarrow = 0$$

If so then the node sets $B(x) = 1$, plays outcome 1, and then halts the stage. If not then the node plays outcome 0.

At stage s where a node $\mathcal{R}_e$ is not acting for the first time, if it has ever played outcome 1 then it plays outcome 1. Otherwise, if it has never acted before, or has acted but never played outcome 1, it checks if

$$\Psi_e(B; x)[s] \downarrow = 0$$

If so then the node sets $A(x) = 1$, plays outcome 1, and then halts the stage. If not then the node plays outcome 0.

At the start of stage s control is given to the $\mathcal{P}_0$ node at the base of the tree and the stage continues until some node halts the stage, or we reach a node of length s, at which point stage $s + 1$ begins. Each node passes control to its immediate successor according to the outcome it plays. So the base node P_0 passes control to its 0 successor when it plays outcome 0, and passes control to its 1 successor when it plays outcome 1.

1.4.2 Verification

First we verify that every stage halts, otherwise we will fail to enumerate infinite sets A, B. Let the *control path* at a given stage be the nodes that receive control during that stage. At stage 0 control is passed to $\mathcal{P}_0$ who performs the actions for a node which receives control for the first time, finishing in halting the stage, as $s = 0$. At stage $s + 1$ control passes through the tree until some node plays outcome 1 for the first time, or the control path reaches length $s + 1$. Clearly then, the control path of stage $s+1$ is bounded in length by $s+1$. Therefore at stage $s+1$ the control path is finite. All computations at stage $s + 1$ are given at most $s + 1$ ticks on the Turing Machine to complete, so all computations at stage $s + 1$ terminate, so every stage terminates.

At no point do we remove any element from A, B so A,B are c.e..

In this tree of strategies outcomes may only change from 0 to 1. In general this may not be the case. So let the *true path* be the path of nodes that are accessed infinitely often. Then we examine requirement $\mathcal{P}_e$ on the true path, call it σ. (There are 2^{2e} copies of $\mathcal{P}_e$, but we are only interested in the one on the true path.) Either

$\sigma * 0$ or $\sigma * 1$ is on the true path. In the former case in for each stage s that $\mathcal{P}_e$ receives control it observes that $\Psi_e(A; x)[s] \uparrow$ or $\Psi_e(A; x)[s] \downarrow \neq 0$, which then leads to it playing outcome 0. In this case $\mathcal{P}_e$ is satisfied as $\Psi_e(A; x) \neq B(x) = 0$. If instead $\sigma * 1$ is on the true path then in some stage s $\mathcal{P}_e$ first plays outcome 1 (and then in every stage beyond that it plays outcome 1). In which case $\Psi_e(A; x)[s] \downarrow = 0$, and $\mathcal{P}_e$ changes $B(x) = 1$ and we observe that $0 = \Psi_e(A; x) \neq B(x) = 1$. As in the first action of a node x is chosen larger than any seen before no module that is a successor of σ can injure σ. With the strict movement from outcome 0 to outcome 1, no module that is acting before σ can injure σ. The situation with $\mathcal{R}_e$ and A is entirely analogous. □

It is easy to see in this construction how the priority ordering is enforced. If a node of higher priority wishes to make a change to A, B then it also changes the control path where new nodes of lower priority come into play. A new node must pick x large to avoid injuring higher priority nodes. In a real sense the tree looks after the priority for us. Of course, Friedberg-Muchnik is a very simple example and moving it to a tree method does not make it much simpler, but it does nicely illustrate the tree priority method.

Chapter 2

The Local Degrees

We shall be focussing on the degrees below $\mathbf{0}'$ which are stratified into the *high/low hierarchy* and the *Ershov hierarchy*.

2.1 The High/Low Hierarchy

The High/Low hierarchy was developed by Cooper [Coo74] and Soare [Soa74] and informally tells us how close to $\mathbf{0}'$ or to $\mathbf{0}$ a degree is.

Definition 1. *Given a set A the* halting problem relative to A *is* $A' = \{e : \Psi_e(A; e)\downarrow\}$. *For a degree* $\mathbf{a}$ *containing A its* jump *is* $\mathbf{a}'$ *equal to the degree of A'. Its $n+1$-th jump is the jump of its n-th jump.*

Thus we see that $\mathbf{0}'$ is the jump of $\mathbf{0}$, and the jump simply relativises the halting problem. Informally, a local degree is high if its jump is as high as possible, and low if its degree is as low as possible. The jump is (non-strictly) order preserving, if $\mathbf{a} \leq \mathbf{b}$ then $\mathbf{a}' \leq \mathbf{b}'$. It is clear that the jump of a local degree must be between $\mathbf{0}'$ and $\mathbf{0}''$, and that the 2-nd jump must be between $\mathbf{0}''$ and $\mathbf{0}'''$, and so on. This gives us the next definition.

Definition 2. *A local degree* $\mathbf{a}$ *is* high_n *if*

$$\mathbf{a}^n = \mathbf{0}^{n+1}$$

A local degree $\mathbf{a}$ *is* low_n *if*

$$\mathbf{a}^n = \mathbf{0}^n$$

A local degree is intermediate *if there is no* $n \in \mathbb{N}$ *for which it is* high_n *or* low_n*.*

$\mathbf{0}$ is uniquely low_0 and $\mathbf{0}'$ is uniquely high_0. If a degree is high_1 or low_1 then we simply call it high or low respectively.

It is natural to ask whether there is always a degree that is high_{n+1} but not high_n, or low_{n+1} but not low_n, for each n. Another natural question is whether intermediate degrees exist. Sacks [Sac63] answered these questions in the positive.

One of the original results in this work will focus on a high_2 degree not having a property that all high degrees have.

2.1.1 Generalised High/Low Hierarchy

One can extend the high/low hierarchy beyond the local degrees. The definition of generalised high/low coincides with the definition of high/low when applied to a local degree.

Definition 3. *The* join *of two degrees* $\mathbf{a}, \mathbf{b}$*, written* $\mathbf{a} \vee \mathbf{b}$ *is the least upper bound of* $\mathbf{a}$ *and* $\mathbf{b}$*. As the set of Turing Degrees is an upper semi-lattice, this always exists.*

The meet *of* $\mathbf{a}, \mathbf{b}$ *is written* $\mathbf{a} \wedge \mathbf{b}$ *and is the greatest lower bound of* $\mathbf{a}, \mathbf{b}$*. This does not always exist.*

A degree $\mathbf{a}$ *is* generalised high_n *if:*

$$\mathbf{a}^n = (\mathbf{a} \vee \mathbf{0}')^n$$

A degree $\mathbf{a}$ *is* generalised low$_n$ *if:*

$$\mathbf{a}^n = (\mathbf{a} \vee \mathbf{0}')^{n-1}$$

A degree is generalised high if it is generalised high$_1$ *and generalised low if it is generalised low*$_1$*.*

2.2 The Ershov Hierarchy

Recall that a set is c.e. iff it is the range of some computable function. We call a degree $\mathbf{a}$ c.e. if there is a c.e. $A \in \mathbf{a}$. One of the original results in this work will focus on the c.e. degrees having a property, but it is useful to know that these form a different stratification of the local degrees.

Definition 4. *A local set* A *is* n.c.e. *if there exists a* Δ_2*-approximating sequence* $(A_s) : s \in \mathbb{N}$ *for* A *for which* $A_s(x) \neq A_{s+1}(x)$ *at most* n *times, and* $A_0 = \emptyset$*. We abbreviate 1.c.e. as c.e..*

Again, this hierarchy does not collapse and it alone does not cover the local degrees. We shall see later that it does cover the local degrees when allowed to extend into transfinite levels.

2.3 Natural Definability

A relationship $R(\bar{x})$ in $\mathcal{D}$ is *definable* (in the language of partial orders) if there is a formula $\phi(\bar{x})$ in the language of partial orders that holds for exactly the tuples $\bar{x}$ such that $R(\bar{x})$ holds.

The problem of natural definability is more hand-wavey. Slaman and Shore [SS99] showed that $\mathbf{0}'$ is definable in $\mathcal{D}$, which relativised to a proof of the definability

of the jump function. Nies, Shore and Slaman [NSS98] used the definability of the jump to prove that all the high/low classes except low are definable. The definability of the class of low degrees remains an open problem. While these results showed that these degrees and classes are definable in $\mathcal{D}$ there was the complaint that the definitions were not natural. They all involve coding models of arithmetic into $\mathcal{D}$, which does not feel "natural". Lewis-Pye [LP12] admits that "natural" is an informal notion but would include the properties that the formula be relatively short and contain few alterations of quantifier. Examples might include $\mathbf{a}$ is the least degree with some simple property, or $\mathbf{b}$ is the greatest degree such that all degrees above have some property. We now give some examples of properties that are unarguably natural.

For two degrees $\mathbf{a}, \mathbf{b}$ we denote their join, or least upper bound, by $\mathbf{a} \vee \mathbf{b}$. Taking $A \in \mathbf{a}$ and $B \in \mathbf{b}$ this is easily seen to always be equal to the degree of $A \oplus B := \{2x : x \in A\} \cup \{2x+1 : x \in B\}$. We denote their meet, or greatest lower bound, by $\mathbf{a} \wedge \mathbf{b}$. Spector [Spe56] proved that this does not always exist.

The following definitions are unarguably natural, but have not yet led to a definition of $\mathbf{0}'$ alone.

Definition 5. *A degree* $\mathbf{a}$ *satisfies the* join *property if for all non-zero* $\mathbf{b} < \mathbf{a}$ *there exists* $\mathbf{c} < \mathbf{a}$ *such that* $\mathbf{b} \vee \mathbf{c} = \mathbf{a}$

A degree $\mathbf{a}$ *satisfies the* meet *property if for all non-zero* $\mathbf{b} < \mathbf{a}$ *there exists* $\mathbf{c} < \mathbf{a}$ *such that* $\mathbf{b} \wedge \mathbf{c} = \mathbf{0}$.

Joining these together we get the complementation property. A degree $\mathbf{a}$ *satisfies the* complementation *property if for all non-zero* $\mathbf{b} < \mathbf{a}$ *there exists* $\mathbf{c} < \mathbf{a}$ *such that* $\mathbf{b} \wedge \mathbf{c} = \mathbf{0}$ *and* $\mathbf{b} \vee \mathbf{c} = \mathbf{a}$.

A degree $\mathbf{m} > \mathbf{0}$ *is minimal if* $\mathbf{b} < \mathbf{m}$ *implies that* $\mathbf{b} = \mathbf{0}$.

Short formulae in the language of partial orders can be given that capture these definitions, so we may call them natural.

If the witness to a property is minimal then we prefix the property with minimal. So for example a degree $\mathbf{a}$ has the minimal meet property if given $\mathbf{b} < \mathbf{a}$ there exists minimal $\mathbf{m} < \mathbf{a}$ such that $\mathbf{m} \wedge \mathbf{b} = \mathbf{0}$.

A few results, presented without proof, in this programme are as follows. Lewis-Pye, Slaman and Seetapun [LP04] established that $\mathbf{0}'$ satisfies minimal complementation. On the other hand, Cooper [Coo89] and independently Slaman and Steel [SS89] established that there are c.e. degrees that do not satisfy join, so do not satisfy the stronger complementation. Epstein [Eps79] conjectured that if $\mathbf{a}$ is a c.e. degree, and $\mathbf{b}$ is an incomputable c.e. degree below $\mathbf{a}$, then there is a minimal degree that complements $\mathbf{b}$ below $\mathbf{a}$. He then proved this for the case where $\mathbf{a}$ is high [Eps81]. The full conjecture was refuted by Cooper and Epstein in 1987 [CE87], but in that same paper it was shown that if $\mathbf{a}$ is low, one can find a minimal degree $\mathbf{m}$ below $\mathbf{a}$ for which $\mathbf{b} \wedge \mathbf{m} = \mathbf{0}$, establishing a weak version of the meet property for $\mathbf{a}$, as it requires $\mathbf{b}$ to be c.e.. It was conjectured in this paper that one can not drop either the assumption that $\mathbf{a}$ is low, or that $\mathbf{b}$ is c.e.. Contradicting an earlier claim by Li and Yang [LY98], Ishmukhametov established that one can drop the requirement that $\mathbf{a}$ be low [Ish03]. We established that every c.e. degree has the meet property, and the proof is presented in chapter 4. This chapter is from joint work with Durrant, Lewis-Pye and Ng [DLPNR16].

Chapter 3

Minimal Degrees

In this chapter we do not present any new results, but we introduce some historical results using methods that we will be using in later chapters. We introduce these methods with these simpler constructions to aid the understanding of the reader. We start with simpler constructions, and add more complexity until we reach the new results.

As we have seen, a degree is minimal if it is non-zero and strictly bounds no non-zero degree. In this chapter we give some historical constructions that establish the existence of minimal degrees, but modified to take place in a priority tree. We first construct a minimal degree using a $\mathbf{0}''$ oracle, then using a $\mathbf{0}'$ oracle, and finally below any c.e. degree using full approximation.

Our requirements for building a set of minimal degree M are always the same - M must be incomputable, and if $\Psi(M) <_T M$ then $\Psi(M) \in \mathbf{0}$. We break these down into the countably infinite list of requirements:

$$\mathcal{P}_e : \Psi_e \neq M$$

$$\mathcal{M}_e : \text{If } \Psi_e(M) \text{ is total then } M \leq_T \Psi_e(M) \text{ or } \Psi_e(M) \text{ is computable.}$$

Thanks to a trick by Posner the method we use to satisfy the $\mathcal{M}_e$ requirements

automatically satisfies the $\mathcal{P}_e$ requirements. The method we use is that of *splitting trees.*

3.1 Splitting Trees

Definition 6. *Two strings σ, τ e-split if $\Psi_e(\sigma)|\Psi_e(\tau)$. If two strings do not e-split then they are e-nonsplitting.*

If a tree T has the property that whenever $\sigma|\tau \in T$ then σ, τ e-split then T is an e-splitting tree. If instead for all $\sigma, \tau \in T$ σ, τ are e-nonsplitting then the tree is an e-nonsplitting tree.

Given a tree T we define the e-splitting subtree above $\sigma \in T$ to be T' where $T'(\lambda) = \sigma$ and $T'(\sigma * i) =$ the left or right member of the first e-splitting found above $T(\sigma)$according to some fixed search procedure, according to whether i is 0 or 1 respectively. This function may be partial, and the tree may be finite or empty.

It is possible for a tree to be neither e-splitting nor e-nonsplitting.

The following pair of lemmas give us the tools we need to satisfy the $\mathcal{M}_e$ requirements. We define a set A to lie on a tree T when there are infinitely many i such that $A \upharpoonright i$ is a string in T.

Lemma 1. *If M lies on a partial-computable e-splitting tree T and $\Psi_e(M)$ is total then $M \leq_T \Psi_e(M)$.*

Proof. We inductively build M from T and an oracle for $\Psi_e(M)$. We know that $T(\lambda) \downarrow$ and is compatible with M so we set $\mu_0 = T(\lambda)$. We then know that one of $T(0)$ and $T(1)$ must be an initial segment of M, so we compute $\Psi_e(T(0))$ and $\Psi_e(T(1))$. One of these computations will converge as $\Psi_e(M)$ is total. As T is e-splitting exactly one of these must be compatible with $\Psi_e(M)$, say it is $T(i)$. As

T is e-splitting, then $T(i)$ is an initial segment of M and we set $\mu_1 = T(i)$.

Inductively, if we have $\mu_s = T(\sigma)$ then we compute $\Psi_e(T(\sigma*0))$ and $\Psi_e(T(\sigma*1))$. As T is e-splitting exactly one of these must be compatible with $\Psi_e(M)$, say $T(\sigma*i)$. We then know that $T(\sigma * i)$ is an initial segment of M, so set it to be μ_{s+1}.

Then $M = \lim_{s\to\infty} \mu_s$, and we have computed M from T and $\Psi_e(M)$. □

Lemma 2. *If M lies on a partial-computable e-nonsplitting tree T then $\Psi_e(M)$ is computable if total.*

Proof. We give an algorithm for computing any given value of $\Psi_e(M; x)$ from T.

Search in some fixed exhaustive manner for $\sigma \in T$ such that $\Psi_e(\sigma; x)\downarrow$. Since T is e-nonsplitting $\Psi_e(\sigma)$ must be compatible with $\Psi_e(M)$, and so $\Psi_e(\sigma; x) = \Psi_e(M; x)$, regardless of σ. □

So if for all e we ensure that M lies on a e-splitting or e-nonsplitting tree we will satisfy the $\mathcal{M}_e$ requirement. We do this by nesting splitting or nonsplitting trees, according to priority. So T_0 will be the Ψ_0 (non)splitting tree, and inside will be T_1 which will be a Ψ_1 (non)splitting tree above some node, inside which will be T_2, and so on. Then M will lie on $T_0, T_1, T_2, \cdots, T_e, \cdots$ and each $\mathcal{M}_e$ requirement will be satisfied. In fact we do not need to build $T_{s+1} \subset T_s$ to be a nonsplitting tree, if above some initial segment of M no strings in T_s $s+1$-split. In this case the complete subtree of T_s above this node is a $s+1$-nonsplitting tree, so we do not need to go to the effort of constructing it.

Thanks to a trick by Posner if we construct M in this manner we do not need separate $\mathcal{P}_e$ requirements. Simply having M on the nested (non)splitting trees ensures that M is incomputable.

Lemma 3. *For a set M, if for each e there is either:*

1. *a partial computable tree T_e that M lies on and is e-splitting,*

2. *an initial segment of M in T_e above which there are no e-splittings in T_e,*

then M is incomputable.

Proof. Assume that M satisfies the conditions in the lemma, and for a contradiction assume that M is computable. Then given a string σ we may computably test whether $\sigma \subset M$. Then Ψ as follows is a Turing functional;

$$\Psi(\sigma; n) = \begin{cases} \sigma(n) & \text{if } \sigma \not\subset M; \\ \uparrow & \text{if } \sigma \subset M. \end{cases}$$

If $\mu \subset M$ then $\Psi(\mu) \uparrow$ so M does not lie on a Ψ-splitting tree. However, every infinite tree must contain Ψ-splittings and any tree that M lies on must be infinite, contradicting the hypothesis that M is computable. □

The previous 3 lemmas give us all we need to build minimal degrees - we need only ensure that M lies on nested e-splitting trees, or for some $e-1$ there is an initial segment of M for which there are no e-splittings above this initial segment in T_{e-1}.

3.2 A minimal degree below $0''$

Thanks to the high strength of the oracle provided there is no need to use a tree priority construction. The proof that follows is essentially due to Epstein [Eps75] following the original proof by Spector.

Theorem 2. *There exists a minimal degree below $\mathbf{0}''$.*

We build a minimal set M using a $\mathbf{0}''$ oracle in our construction. Thanks to our lemmas we have one set of requirements to satisfy:

$$\mathcal{M}_e : M \text{ lies on a } e\text{-splitting or } e\text{-nonsplitting tree.}$$

The requirements are prioritised as follows:

$$\mathcal{M}_0 <_L \mathcal{M}_1 <_L \mathcal{M}_2 <_L \cdots <_L \mathcal{M}_e <_L \cdots$$

We define a sequence of perfect computable trees $\{T_s : s \in \omega\}$ with $T_s \supset T_{s+1}$, and a sequence of finite strings $\{(\mu_s) : s \in \omega\}$ such that $\mu_s \subset \mu_{s+1}$. For each s we require that μ_s lies on T_s and that T_s is s-splitting or that μ_s has no s-splitting extensions in T_s. The set $M = \cup_{s\in\omega}\mu_s$ then satisfies our requirements.

3.2.1 Construction

At stage 0 set $\mu_0 = \lambda$ and $T_0 = 2^{<\omega}$. (As Ψ_0 is the identity functional, T_0 is indeed the complete 0-splitting tree above λ.)

At stage $s+1$ we inductively have $T_0 \supset \cdots \supset T_s$ and $\mu_s \in T_s$. If all extensions of μ_s in T_s have $s+1$-splitting extensions then let T_{s+1} be the $s+1$-splitting subtree of T_s above μ_s and let μ_{s+1} be the left successor (arbitrarily) of μ_s in T_{s+1}. If there is instead some extension of μ_s in T_s that has no $s+1$-splitting extensions then let μ_{s+1} be the least such and let T_{s+1} be the complete subtree of T_s above μ_{s+1}.

3.2.2 Verification

It is clear that the requirement $\mathcal{M}_s$ is satisfied at stage s as we either build the splitting or nonsplitting tree entirely in this stage. As the requirements $\{\mathcal{M}_e : e \in \mathbb{N}\}$ are satisfied, we can call on the lemmas to show that M is minimal.

It only remains to establish the strength of oracle used during the construction. We asked whether all extensions of a string have e-splitting extensions, which is seen to be a Π_2-question, i.e. it is "For all extensions of a string, there is a pair of strings σ, τ and a stage s that extend this where $\Psi(\sigma)[s]|\Psi(\tau)[s]$. Π_2 questions are answerable with a $\mathbf{0}''$ oracle. As $\mathbf{0}''$ is not itself minimal, M must be strictly below $\mathbf{0}''$.

3.2.3 Remarks

During the construction it is arbitrary whether we take the left or right successor of μ_s. We can (and will) use this freedom to add extra requirements to the minimality requirements, such as diagonalising against a given set. This observation also means that we can modify this construction to take both the left and right branches and find $\aleph_0$ many incompatible minimal degrees below $\mathbf{0}''$.

3.3 A Minimal Degree Below $\mathbf{0}'$

Theorem 3. *There exists a minimal degree below* $\mathbf{0}'$

This time we do not get to ask any Π_2 questions as we do not have an oracle as strong as $\mathbf{0}''$. As it was this oracle that gave us the strength to build the trees in one stage we no longer build a tree in one stage, but instead add one branch at a time. We can, however, ask the weaker question "Is there a e-splitting extension of μ_s?". If so then we search in a computable manner for this e-splitting and enumerate it into our tree, keeping the tree computably enumerable. We work within this limitation by guessing that if there is one more e-splitting that we can enumerate into T_e then there will always be another e-splitting that we can enumerate into T_e. In the limit this guess may be validated, or there will be some stage where we

go from enumerating e-splittings into T_e to declaring T_e to be a nonsplitting tree above some initial segment. However, this does mean that in this construction we will have partial trees.

The requirements and the priority ordering are identical to the case for $\mathbf{0}''$. This time we place the requirements on a tree of strategies, not out of necessity, but out of a desire to demonstrate the techniques we will be using. Requirement $\mathcal{M}_0$ will lie at the base of the tree, and requirement $\mathcal{M}_e$ will have two outcomes $0 <_L 1$ which will both have a $\mathcal{M}_{e+1}$ requirement at the end of them. Outcome 1 will occur when the node finds e-splittings to enumerate into T_e, and outcome 0 will occur when there are no e-splittings to enumerate into T_e.

It would be pointless to try to build $T_e \subset T_{e-1}$ if T_{e-1} is going to be finitary. So for a node σ we shall denote σ^* to be the greatest $\tau \subset \sigma$ such that $\tau * 1 \subset \sigma$.

At stage $s+1$ there will be two phases. One where the trees that are believed to be e-splitting will be extended, and a second where μ_s is defined. In the first phase, control is passed to $\mathcal{M}_\lambda$ and modules act according to the outcomes played by its immediate successor node. The phase will end when the control path reaches length $s+1$. A $\mathcal{M}_e$ node, σ, that is given control will ask $0'$ if there is an e-splitting extension of μ_s among strings in T_{σ^*}. If there is then the node will declare T_e to be the e-splitting subtree of T_{σ^*} and perform a computable search for such a pair, and enumerate this pair into T_e, and play outcome 1. Otherwise the node will play outcome 0.

In the second phase μ_{s+1} will initially be set to λ. For each node σ that played outcome 1 in the first phase, μ_{s+1} will be extended to be the immediate successor of μ_{s+1} on T_σ.

3.3.1 Construction

At stage 0 λ receives control for the first time, in phase 1 sets $T_\lambda(\tau) = \tau$ for all strings τ, and in phase 2 sets $\mu_0 = \lambda$. Then the stage is halted.

At stage $s+1$, phase 1, control is passed to λ and the stage continues until the control path is of length $s+1$.

A node σ that is passed control for the first time at stage $s+1$ declares $T_\sigma(\lambda)$ to be μ_s. It asks if there is an e-splitting extension of μ_s among strings in T_{σ^*}. If not, then σ declares T_σ to be the complete subtree of T_{σ^*} above μ_s and plays outcome 0. If so, then σ declares T_σ to be the e-splitting subtree of T_{σ^*} above μ_s and performs a computable search for an e-splitting above μ_s among strings in T_{σ^*} to enumerate into T_σ. It then plays outcome 1.

A node σ, that is a $\mathcal{M}_e$ node, that has ever played outcome 0 continues its definition of T_σ as a complete subtree. If σ has never played outcome 0 then it asks $\mathbf{0}'$ if there is an e-splitting above μ_s among strings in T_{σ^*}. If so then it continues the definition of T_σ as an e-splitting subtree, and performs a computable search for another pair to enumerate into this tree, and plays outcome 1. Else, it declares T_σ to be the complete subtree of T_{σ^*} above μ_s and plays outcome 0.

In phase 2, μ_{s+1} is initially defined to be λ. For each node σ that played outcome 1 in phase 1, μ_{s+1} is extended to be the immediate successor along the left path of T_σ.

3.3.2 Verification

In this construction it is not as simple as declaring that at stage s requirement $\mathcal{M}_s$ is satisfied. As a node that ever plays outcome 0 will play outcome 0 in any stage it receives control, we may let the true path be the path of nodes that receive control

in infinitely many stages, as this is uniquely defined. Each node performs a standard search for e-splitting strings, but only in the case where they are known to exist. Therefore each node will pass control. The length of the control path is bounded by the stage, so each stage will halt.

μ_s is well-defined in each stage, and we can see that $M = \lim_{s\to\infty} \mu_s$ is well-defined.

To see that requirement $\mathcal{M}_e$ is satisfied we must look at which node σ on the true path is of level e, and which outcome it plays on the true path. If $\sigma * 0$ is on the true path, then in some stage $s+1$, $\mathbf{0}'$ told σ that there is no e-splitting extension of μ_s among strings in T_{σ^*}. In this case σ correctly identifies the complete subtree of T_{σ^*} above μ_s as a e-nonsplitting tree. It is also clear that $\mu_s \subset M$, so M lies on T_σ. Else, if $\sigma * 1$ is on the true path, then in the limit T_σ is built as a e-splitting tree which M lies on. In either case, the requirement is satisfied.

We only ask a Σ_1 question in establishing whether there is an e-splitting among a given collection of strings, which is answerable by a $\mathbf{0}'$ oracle. Again $\mathbf{0}'$ is not minimal so $M < \mathbf{0}'$ is minimal.

3.3.3 Discussion

It is not even necessary to ask $\mathbf{0}'$ whether there is an e-splitting among strings in T_{σ^*}. We can run a recursive search for e-splittings, bounding the search in each stage. Of course, we may slow down the construction as we wait for a computation to converge that will never converge, rather than having $\mathbf{0}'$ tell us immediately whether it converges or not, but it does mean that we can push down further on what bounds a minimal degree.

Traditionally this construction involves redefining T_e finitely many times. We

have had the priority tree take care of this detail for us by having 2^e copies of T_e floating around in potential, but only one of them lying on the true path.

3.4 A Minimal Degree Below an Incomputable c.e. Degree

This construction is originally due to Yates [Yat70], but is heavily modified here to use the tree framework.

As we do not need to ask $\mathbf{0}'$ whether there is an e-splitting, and instead may just search for them, we can push the sets that bound minimals further downwards. However, if all we do is drop the oracle question then all we achieve is a Δ_2 minimal degree, which is just a minimal degree below $\mathbf{0}'$. To actually push the minimals further down the hierarchy we need to deal with *permission*. Therefore, the result of this section is:

Theorem 4. *Every incomputable c.e. degree bounds a minimal degree.*

Let A be an incomputable c.e. degree, with c.e. sequence $(A_s)_{s\in\mathbb{N}}$. The rough idea with creating M with permission from A is that for M to change below some bound, A must first change below that bound, i.e. to change $M(x)$ in stage t we want to see a change in $A(x')$ where $x' \geq x$, and we want to see this in stage $s > t$. In this way if we wrote axioms for a functional that computed M from A the axioms would be consistent. In this case we need to ensure that introducing this delay in enumerating e-splittings into our trees does not incorrectly turn a e-splitting tree into an e-nonsplitting tree. In this case we will be able to argue that if this were to happen then we would have an algorithm for computing A. For if we found e-splittings always after our approximation to A stabilised then we could

simultaneously look for e-splittings and approximate A. Whenever we find a greater e-splitting (say of height n) then we can declare $A \upharpoonright n$ to be correct, and enumerate larger and larger initial segments of A in this manner.

Apart from adding permission and removing the $\mathbf{0}'$ oracle this construction is largely the same as the previous construction. The tree of strategies remains the same, but in addition to the node σ containing a tree T_σ it contains a collection of tuples $t(\sigma) = ((\tau_0, \tau_1), \cdots, (\tau_l, \tau_m))$ of e-splitting pairs that are awaiting permission to be enumerated into T_σ. We will consider these tuples to be ordered by the length of the shortest element.

In this construction, a module that has previously played outcome 0 may later play outcome 1, as it reaches the required length of time to find an e-splitting. Therefore, we shall take the True Path to be the *leftmost* path of nodes that receive control in infinitely many stages.

3.4.1 Construction

At stage 0 control is passed to λ which sets $\mu_0 = \lambda$ and $T_\lambda(\tau) = \tau$, for all strings τ. Stage 0 is then halted and stage 1 begins.

In phase 1, if a node σ receives control at stage $s+1$ for the first time then it sets $T_\sigma(\lambda) = \mu_s$ and halts the stage. Else, if the control path is of length $s+1$ the stage is halted.

In phase 1, if a node σ receives control at stage $s+1$ for a subsequent time then if $t(\sigma)$ is non-empty σ checks if $A_s \upharpoonright |\tau_i \wedge \tau_{i+1}| \neq A_{s+1} \upharpoonright |\tau_i \wedge \tau_{i+1}|$ for some shortest $(\tau_i \wedge \tau_{i+1})$ in $t(\sigma)$. If so then enumerate τ_0, τ_1 into T_σ, empty $t(\sigma)$, and play outcome 0. Otherwise $t(\sigma)$ is empty or the condition fails, in which case σ searches for an e-splitting (τ_0, τ_1) greater than any e-splitting in $t(\sigma)$ extending μ_s among strings in

T_{σ^*}, where the meet of τ_0 and τ_1 is not the meet of any tuple in $t(\sigma)$. If one is found then σ enumerates them into $t(\sigma)$. Then σ plays outcome 1.

Define μ_{s+1} in phase 2 exactly the same as in the previous construction: initially define $\mu_{s+1} = \lambda$, and extend it among immediate successors in trees of nodes that played outcome 0.

3.4.2 Verification

We are working without an oracle, and clearly every node performs only computable actions, with bounded searches. Therefore no node enters an infinite loop. The length of the control path in stage $s+1$ is bounded by length $s+1$ so clearly every stage halts.

We let the true path be the leftmost path of nodes that receive control in infinitely many stages.

We need to establish that adding A-permission will not change a true path 0 outcome to a true path 1 outcome. Assume for contradiction that σ lies on the true path and $\sigma * 1$ lies on the true path despite in infinitely many stages σ finds greater e-splittings among strings in T_{σ^*}. In this case we have an algorithm for computing A using the machine that computes M. Whenever node σ enumerates a pair of strings μ_0, μ_1 into $t(\sigma)$ at stage s then we know that $m = \min(|\mu_0|, |\mu_1|)$ satisfies $A_s \upharpoonright m = A \upharpoonright m$. So to compute $A(x)$ we simply wait for $m > x$ such that $A_s \upharpoonright m = A \upharpoonright m$ and read off $A_s(x)$. But we assumed that A is incomputable, giving us the required contradiction.

To see that requirement M_e is satisfied, again let σ be of level e on the true path. If $\sigma * 0$ is on the true path then we successfully enumerate greater and greater e-splittings into T_σ and we ensure that M lies on T_σ, so M_e is satisfied. If $\sigma * 1$ lies

on the true path then we have seen that it can not be the case that $|t(\sigma)| \to \infty$ so for some stage T_{σ^*} above μ_s is an e-nonsplitting tree, and we again satisfy M_e.

To see that M is A-computable we can write axioms at the end of stage s of the form $\Psi(A_s) = \mu_s$. As changing M below x requires first observing A to change below x these axioms will be consistent and we see that as A_s tends to A, μ_s tends to M and the axioms compute M from A.

3.5 A Minimal Degree Below a Generalised High Degree

Cooper [Coo71] established that every high degree bounds a minimal degree, a result that was strengthened by Jockush [JJ77] to every generalised high degree bounds a minimal degree. We give the latter result because it is more general, despite it taking us outside the local degrees.

Theorem 5. *Every generalised high degree bounds a minimal degree.*

A generalised high degree $\mathbf{a}$ has the property that $\mathbf{a}' = (\mathbf{a} \cup \mathbf{0}')'$, and $\mathbf{a}'$ is the greatest degree computably enumerable in $\mathbf{a}$. So $\mathbf{a}$ can approximate $(\mathbf{a} \cup \mathbf{0}')'$, in that there is a function $f \leq_T \mathbf{a}$ such that $\lim_{s\to\infty} f(x, s) = 1$ iff $x \in (\mathbf{a} \cup \mathbf{0}')'$. This makes the construction of a minimal degree below a generalised high degree similar to the construction below $\mathbf{0}'$ and so we give a construction that does not use a tree of strategies.

So we may ask the question "Is there an e-splitting above every initial segment of μ among strings in W_j". Let $f \leq_T \mathbf{a}$ be the function that approximates the answer to this question, i.e. $\lim_{s\to\infty} f(e, j, s) = 1$ if there is an e splitting above every initial segment of μ among strings in W_j, and 0 otherwise. Now we search to extend a tree

T_e whenever f gives us evidence that it will be an e-splitting tree, and we refuse to extend T_e whenever f suggests that it will be a e-nonsplitting tree. f is correct in the limit, so beyond some stage it must be correct.

3.5.1 Construction

At stage 0 define T_0^0 to be the identity tree $2^{<\omega} \to 2^{<\omega}$ and $\mu_0 = \lambda$.

At stage $s+1$ we act in two phases. In phase 1 we have a list of trees

$$T_0^s \supset T_1^s \supset \cdots \supset T_k^s$$

together with an index i_j that defines T_j as a c.e. set of strings W_{i_j}. We shall identify i_j with T_j. Firstly we need to check that we did not stop enumerating strings into a tree too rashly. Define a tree T_e for each $e < k$ to *require attention* at stage $s' > s+1$ if T_e was being built as a subtree of $T_{e'}$, $f(e, e', s) = 0$ and some fixed search procedure finds an e-splitting among strings in T_e extending μ_t, where t is the first stage T_e was defined, in less than $s+1$ stages. For the least e such that T_e requires attention perform the following;

1. Define T_e^{s+1} to be the e-splitting subtree of $T_{e'}$ above μ_s.

2. Define T_j^{s+1} to be T_j^s for $j < e$.

3. Make T_j^{s+1} undefined for $j > e$.

4. Declare that T_e has *received attention*.

5. Halt the phase.

If no T_e required attention then for each e such that T_e is defined to be the e-splitting subtree of $T_{e'}$ then search for the least $t > s+1$ such that either:

1. An e-splitting above μ_s is found among strings in $T_{e'}$ in fewer than t stages.

2. $f(e, e', t) = 0$

If case 1 applies then we enumerate this e-splitting into T_e^{s+1}. If case 2 applies then we define T_e to be the total subtree of $T_{e'}$ above μ_s. Phase 1 is then halted, and phase 2 begins.

In phase 2 define μ_{s+1} to be a leaf of the last tree that is defined to be an e-splitting tree. Finally halt the stage.

3.5.2 Verification

We verify that eventually every tree is accurately identified as a e-splitting or e-nonsplitting tree, by induction. For the base step T_0 is immediately defined to be $2^{<\omega}$ which is indeed the total 0-splitting tree. Now inductively assume that T_f is correctly identified as the f-splitting or f-nonsplitting tree for $f < e$. If T_e is identified as the e-nonsplitting tree in some stage, and T_e never receives attention, then T_e was accurately identified as a e-nonsplitting tree. If T_e does receive attention then at some stage f must change its mind about T_e being a nonsplitting tree, or T_e receives attention in finitely many stages. If, beyond some stage, f identifies T_e as a e-splitting tree, then f must be correct, as it is correct in the limit.

Chapter 4

Every c.e. Degree has the Meet Property

This is joint work with Durrant, Lewis-Pye and Ng [DLPNR16].

Theorem 6. *Given an incomputable c.e. degree* $\mathbf{a}$ *and any degree* $\mathbf{b} < \mathbf{a}$ *there is a minimal degree* $\mathbf{m} < \mathbf{a}$ *such that* $\mathbf{m} \not\leq \mathbf{b}$

Corollary 1. *Every c.e. degree has the (minimal) meet property.*

4.1 Requirements & Notation

Given that $\mathbf{a}$ is c.e., we take A to be a c.e. set in $\mathbf{a}$, with c.e. approximating sequence $\{A_s : s \in \mathbb{N}\}$. Since we have $\mathbf{b} < \mathbf{a}$ we take $B \in \mathbf{b}$ and a Turing functional Γ such that $B = \Gamma^A$. By speeding up the enumeration of A as necessary, we let $\{B_s\}_{s\in\mathbb{N}}$ be a computable approximation of B, with B_s being a finite binary string of length s such that $B_s \subseteq \Gamma^{A_s}$. While B_s is considered to be a finite binary string, we consider A_s to be an infinite string, by having $A_s(x) = 0$ for any x greater than the greatest value enumerated into A_s.

Given these, we must construct M of minimal degree below $\mathbf{a}$ avoiding the lower cone of $\mathbf{b}$. We build M by specifying a computable approximation $\{\mu_s\}_{s\in\mathbb{N}}$ where each μ_s is a finite binary string, and let $M = \lim_s \mu_s$. We have two classes of requirements - forcing M to be minimal, and forcing M to avoid the lower cone of $\mathbf{b}$. We split this into two countably infinite sets of requirements $\{\mathcal{P}_e : e \in \mathbb{N}\}$ and $\{\mathcal{M}_e : e \in \mathbb{N}\}$ such that requirement $\mathcal{P}_e$ diagonalises M against $\Psi_e(B)$, if the evidence is that $\Psi_e(B)$ is total. Requirement $\mathcal{M}_e$ ensures that $\Psi_e(M)$ is either computable, or computes M, by constructing nested splitting trees.

4.1.1 The Construction Tree

We work on a construction tree that is a subtree of $(\omega + 1)^\omega$. Each node of length $2e$ is assigned the requirement $\mathcal{M}_e$, with two outcomes $\infty <_L f$. The outcome ∞ indicates that the node believes that the splitting tree it is constructing is infinite, and the outcome f indicates that the node believes that above some initial segment of M there are no permissible e-splittings. Nodes of length $2e + 1$ are given the requirement $\mathcal{P}_e$ with $\omega + 1$-many outcomes labelled $0 <_L 1 <_L 2 <_L \cdots <_L f$. Outcome f will occur when there is some argument m for which $\Psi_e(B; m)\uparrow$, or else the node successfully diagonalises M against $\Psi_e(B)$, and hence only requires finite action. Each of the other outcomes can be thought of as a guess as to the least m for which there are infinitely many s with $\Psi_e(B_s; m)\downarrow$ with different uses, and hence the observed uses are unbounded. In this case the use of $\Psi_e(B; m)$ is not finite, so $\Psi_e(B; m)\uparrow$ and the requirement is satisfied.

For nodes on the construction tree σ, τ we write $\sigma <_L \tau$ to say that σ is strictly to the left of τ. We then consider the nodes to be ordered lexicographically, so σ has higher priority than τ if $\sigma <_L \tau$ or $\sigma \subset \tau$.

At stage s we define TP_s, which is our s-th approximation to the true path, which indicates the s nodes visited by the construction at stage s.

If a node σ on the construction tree is assigned the requirement $\mathcal{M}_e$ or $\mathcal{P}_e$ then we shall write Ψ_σ for Ψ_e.

4.2 Outline of the Proof

4.2.1 Ensuring $M \leq_T A$

This requirement does not get assigned a node on the tree of strategies. Instead, each stage is broken into multiple phases, and this requirement is assigned one of the phases.

We shall enumerate axioms for a functional Φ such that $\Phi(A) = M$. These axioms will be enumerated at the end of each state s, for arguments $n < s$. The use for this computation will be as follows. Let s^* be the maximum of n and $s' - 1$, where s' is a stage $\leq s$ where a number $\leq n$ has been enumerated into A. Let α be the least initial segment of A_s such that $B_s \upharpoonright s^* \subseteq \Gamma(\alpha)$. Then at the end of stage s, we define $\Phi(\alpha; n) = \mu_s(n)$.

At any point during stage s we say that α is permissible if it is compatible with $\Phi(A_s)$.

4.2.2 Satisfying $\mathcal{M}_e$

This requirement is satisfied in the same way as we saw in chapter 3.4, a minimal degree below an incomputable c.e. degree.

For a node σ, that is not the base node, we let σ^* be the greatest $\tau \subset \sigma$ that is a $\mathcal{M}$ node such that $\tau * \infty \subset \sigma$. Since we assume that Ψ_0 is the identity functional,

the base node will always play outcome ∞, so σ^* is always well-defined. Then T_σ will be built as a subtree of T_{σ^*}.

4.2.3 Satisfying $\mathcal{P}_e$

We would like to use the standard technique of diagonalisation: choose a value x and wait until $\Psi_e(B)(x) = M(x) = 0$, and then change $M(x)$ to 1. Unfortunately, this simple technique does not work because we need to coordinate the diagonalisation with A-permission. Instead, we can use the fact that $A \not\leq_T B$ to wait for A-permission.

Suppose we monitor $\Psi_e(B) \upharpoonright n$ for a fixed n. If we find that $\Psi_e(B) \upharpoonright n$ agrees with $\mu_s \upharpoonright n$, then we would like to modify $\mu_s \upharpoonright n$, and could do so with a suitably low A-change, then we would successfully diagonalise. While waiting for a suitable A-change we can map the initial segment of B that computes μ_s to the initial segment of A that we are waiting to change. Then we can look at a bigger n. If we never get an appropriate A-change then we map all of B to all of A, contrary to $A \not\leq_T B$, so we must eventually get an appropriate A-change to diagonalise, or we already have $M \upharpoonright n$ not an initial segment of $\Psi_e(B)$.

That is how the $\mathcal{P}_e$ nodes would work if there were no $\mathcal{M}_e$ nodes. Since we need to coordinate both classes of nodes on the tree of strategies, we need to add more complexity. The $\mathcal{P}_e$ module σ will build a functional Ψ_σ that will try to compute A from B. It will do this by containing ω-many modules $N^0_\sigma, N^1_\sigma, \cdots$. The module N^i_σ is responsible for enumerating axioms for $\Psi_\sigma(i)$, where we try to make $\Psi_\sigma(B; i) = A(i)$. For convenience, we build Ψ_σ as a c.e. set of strings, and ensure that if $i \in A$ then after i is enumerated into A no more strings are enumerated into $\Psi_\sigma(i)$. Then to compute $\Psi_\sigma(X; i)$ one runs the enumeration of $\Psi_\sigma(i)$ until either τ

is found that $\tau \subset X$, or else i enters A. In the former case, output 0, in the latter case, output 1.

While $i \notin A_s$ module N^i_α waits until it sees a string $\nu \subseteq \Psi_e(\tau)$ for some $\tau \subseteq B_s$ and $\nu \subseteq \mu_s$ where $\nu = T_{\alpha^*}(\rho)$ for some ρ specific to the module. Then the module enumerates τ into $\Psi_\sigma(i)$ as well as the *demand* (τ, i, ν_0, ν_1) where $\nu_0 = T_{\alpha^*}(\rho^-)$ and $\nu_1 = T_{\alpha^*}(\rho^\dagger)$. This demand should be read as "If $\tau \subset B$ and $i \in A$ then if $\nu_0 \subset M$ then $\nu_1 \subset M$.

Some demands will be acted on, and some will be ignored at stage s. When one is acted upon then we say it is *implemented* at stage s. This will be defined precisely in the formal construction.

We would have preferred to issue demands of a simpler form - "if $\tau \subset B$ and $i \in A$ then $\nu_a \subset M$" , or "if $\tau \subset B$ and $i \in A$, then $\nu \not\subset M$." We can not do this because of trickiness involving the tree of strategies. For A-permission to be effective, nodes to the left of TP_s are required to be able to issue demands at stage s. The problem with the first of the proposed simplifications is that if $i \in A$ then we will have permission to change our mind as to whether $\nu_s \subseteq \mu_s$ whenever we see a change in $\tau \subseteq B_s$ as long as μ_0 is an initial segment of μ_s. The second alternative leads to more subtle problems with the interactions between $\mathcal{P}$ requirements.

4.3 Formal Construction

4.3.1 Initialisation

First, let us define what it means for a node σ on the construction tree to be initialised, and when this takes place.

If σ is a $\mathcal{M}_e$ node then to initialise it we make $T_\alpha(\rho) \uparrow$ for all ρ. We also discard

all splittings found. If σ is a $\mathcal{P}_e$ node then we discard all axioms enumerated for Ψ_σ as well as all demands issued by modules of σ. We also discard all *recorded computations* for σ, which will be defined later in the construction.

For either type of node we make z_σ undefined, so that z_σ can be a number chosen to be large every time σ is first visited after being initialised. This will help make sure that nodes don't interfere with nodes of higher priority.

The conditions that will make σ be initialised are independent of its type. At stage s the node σ is initialised as soon as any of the following conditions are met:

1. $s = 0$.

2. A node strictly to the left of σ is visited.

3. τ enumerates strings into T_τ at stage s, where $\tau * f \subseteq \sigma$, or $\tau <_L \sigma$.

4. A demand issued by a module N^j_τ such that $\tau <_L \sigma$ or $\tau * i \subseteq \sigma$ for $j < i$ is implemented at stage s, but was not implemented at stage $s-1$, or vice-versa.

At any point of any stage, a module is *active* if it has been visited subsequent to its last initialisation. We call a tree T_σ active if σ is active.

4.3.2 Phases of stage s

If $s = 0$ then all nodes are initialised, and the stage halts. If $s > 0$ the instructions are broken into 4 phases, where the 3rd is where we visit nodes on the construction tree.

1. This is the phase of tree enumeration. For each σ that is a $\mathcal{M}$ node, in order of priority, consider σ's list of splittings. If there is a first that is permissible then enumerate it into T_σ and empty σ's list of splittings.

2. This is the phase of defining μ_s. We perform the following iteration, terminating after a finite number of steps. At each step we redefine the string μ_s^*, initially the empty string, and finally μ_s takes the final value of μ_s^*. The iteration defines a path through the nested splitting trees, taking the left or right path according to issued demands, and taking the left path if no demand exists. As we proceed we enumerate pairs of the form (μ, σ) to keep track of the priority at which we have implemented demands.

   At step 0, define $\mu_s^* = \lambda$. At step $k > 0$ check to see if there is a demand issued by some module N_σ^i of the form (τ, i, ν_0, ν_1) such that $\tau \subseteq B_s, i \in A_s, \nu_0 \subseteq \mu_s^*$ and we have not yet enumerated a pair (μ, τ) during this iteration with $\mu \supset \nu_0$ and τ of higher priority than β. If so, choose the demand such that ν_0 is the shortest, and declare this demand implemented at stage s, redefine $\mu_s^* = \nu_1$, enumerate the pair (ν_1, σ) and go to the next step. This simply finds the demand of highest priority that desires to be implemented, and implements it. If there is no demand that needs implementing, check to see if there is σ such that $\mu_s^* \in T_\sigma$ but is not a T_σ-leaf. If there is, redefine μ_s^* to be the left successor of μ_s^* on T_σ and go to the next step of the iteration. Otherwise define μ_s to be μ_s^*.

   Note that a demand may be implemented, then injured by a demand of higher priority. Say node σ implements a demand causing $\mu_\sigma \subset \mu_s^*$, and a node τ of higher priority has a demand of the form $(\mu_\sigma, *, *, \mu_\tau)$, where $\mu_\tau | \mu_\sigma$. Then $\mu_\tau \subset \mu_s$, not μ_σ.

   As at stage s there are only finitely many demands in existence, and only finitely many splitting trees with finitely many elements, the iteration must terminate.

3. This is the priority tree visiting phase. We define TP_s, the nodes visited at stage s. Let $\sigma = TP_s \upharpoonright i$ be defined. We describe the actions taken by σ and decide the outcome played. If $|\sigma| \geq s$ then σ performs no actions in this stage, and we terminate phase 3 of stage s. If z_σ is undefined then we choose it to be a large odd number. Then we act according to what type of node σ is.

   If σ is assigned requirement $\mathcal{M}_e$, then if $T_\sigma(\lambda) \uparrow$ then we set $T_\sigma(\lambda) = T_{\sigma^*}(\rho)$ where $|\rho| = z_\sigma$, and $T_{\sigma^*}(\rho) \subseteq \mu_s$. If ρ does not exist then we leave $T_\sigma(\lambda) \uparrow$. Otherwise, if $\nu \subset \mu_s$ for some T_σ-leaf ν, we search for Ψ_e-splittings above ν of length $\leq s$ consisting of strings of odd level on T_{σ^*}, and enumerate any found into σ's list of splittings. If strings have been enumerated into T_σ since the last stage σ was visited, or σ is the base node λ, then σ plays outcome ∞. Otherwise, σ plays outcome f.

   If σ is assigned requirement $\mathcal{P}_e$, then we find the least $i < s$ that *requires attention*, if any. This is the case if there is $\mu \subseteq \mu_s$ such that $\mu = T_{\sigma^*}(\rho)$ for ρ of length $p^i_{z_\sigma}$, and $\mu \subseteq \Psi_e(\tau)$ for some shortest $\tau \subseteq B_s$, but N^i_σ has not yet 'recorded the computation Ψ^τ_e. If no N^i_σ requires attention then σ performs no action and plays outcome f. Otherwise, let i be the least such that N^i_σ requires attention. Declare $\Psi_e(\tau)$ to be a recorded computation. If $i \notin A$ then issue the demand (τ, i, ν_0, ν_1), where ρ is as above, $\nu_0 = T_{\sigma^*}(\rho^-)$ and $\nu_1 = T_{\sigma^*}(\rho^\dagger)$, and enumerate τ into $\Psi_\sigma(i)$. Then σ plays outcome i.

4. This is the phase where we define Φ. For each $n < s$ where $\mu_s(n) \downarrow$, let s_0 be the maximum of n and $s_1 - 1$, where s_1 is a stage $\leq s$ at which a number $\leq n$ was enumerated into A. Let α be the shortest initial segment of A_s such that $B_s \upharpoonright (s_0 + 1) \subseteq \Gamma^\alpha$. Define $\Phi(\alpha; n) = \mu_s(n)$.

4.4 Verification

First we must verify that the instructions are well defined. The only part that is not obvious is during phase 2 of stage s where the instructions for step $k > 0$ require us to select the demand (τ, i, ν_0, ν_1) for which ν_0 is shortest. We must ensure that there is a unique such demand. After this has been verified it is clear that the instructions for each phase of each stage are finite since:

1. At stage s, if the demand (τ, i, ν_0, ν_1) is implemented at step k of the iteration, and (τ', k, ν_2, ν_3) is implemented at step $k' > k$ then $\nu_2 \supset \nu_0$.

2. At each stage, only finitely many demands are issued, and only finitely many strings are enumerated into trees.

So we wish to ensure that at any point of the construction, if demands (τ, i, ν_0, ν_1) and (τ', k, ν_2, ν_3) are both issued, and not discarded by initialisation, then $\nu_0 = \nu_2$ implies $i = j$ and both demands were issued by the same module M^i_σ.

Since z_σ is chosen to be large whenever a node is visited for the firs time after an initialisation, it follows that when $\mu \in T_\sigma \cap T_{\sigma'}$ for distinct nodes σ and σ', we must have $\sigma * \infty \subset \sigma'$ or vice-versa. That is, when a string belongs to two valid trees, it must be the case that one of these trees was built as a subtree of the other. Then the following three facts combine to give the requirement:

1. If σ is a $\mathcal{M}$ node, then strings of T_σ are of odd level in T_{σ^*}

2. If N^i_σ issues a demand (τ, i, ν_0, ν_1) then ν_0 is of even level in T_{σ^*}.

3. If σ_1, σ_2 are $\mathcal{P}$ nodes, and $T_{\sigma_1^*} = T_{\sigma_2^*}$ then since $z_{\sigma_1} \neq z_{\sigma_2}$, if we have demands (τ, i, ν_0, ν_1) and (τ', k, ν_2, ν_3) issued by $M^i_{\sigma_1}$ and $M^j_{\sigma_2}$ respectively, we must have $\nu_0 \neq \nu_2$.

Therefore, the construction is well defined, and the instructions at each stage are finite.

Lemma 4. *At every stage s, μ_s is permissible. M is total and $\Phi(A) = M$.*

Proof. As A is incomputable, it follows that T_λ is infinite. As every tree is ultimately built as a subtree of T_λ it follows that for every length l there is a stage s with $|\mu_s| > l$. In the construction of Φ the use on argument n is clearly bounded, so the second statement of the lemma follows from the first. So we prove the first.

We proceed by induction on s. Since $\mu_0 = \lambda$, the lemma holds for stage 0. If μ_s is compatible with μ_{s-1}, and μ_{s-1} was permissible, then μ_s is permissible. So assume that $\mu_s | \mu_{s-1}$, and consider the iteration that takes place during phase 2 of stages s and $s-1$. At each step of the iteration, either a demand is implemented, or we find σ of lowest priority where $\mu_s^* \in T_\sigma$, but is not a T_σ leaf, and extend μ_s^* to be the left successor in T_σ, or we terminate the iteration. Therefore there must be a step k where the iterations at stages s and $s-1$ diverge. There are 3 possibilities:

1. Step k at stage $s-1$ implements a demand (τ, i, ν_0, ν_1) and in step k of stage s it is not the case that a demand (τ', k, ν_2, ν_3) with $\nu_2 \subset \nu_0$ is implemented. In this case we see that i was enumerated into A at a stage $> |\tau|$, and since ν_0 is of length $> i$, any $\alpha \subset A_{s-1}$ such that $\nu_0 \subseteq \Phi(\alpha)$ at the end of stage $s-1$, is sufficiently long that $\tau \subseteq \Gamma(\alpha)$. Since the demand (τ, i, ν_0, ν_1) is not implemented at stage s, $\tau \not\subseteq B_s$, and so any extension of ν_0 is permissible.

2. At step k of stage s a demand (τ, i, ν_0, ν_1) is implemented, but during step k of stage $s-1$ no demand (τ', j, ν_2, ν_3) is implemented with $\nu_2 \subseteq \nu_0$. In this case, as the demand (τ, i, ν_0, ν_1) was not implemented at stage $s-1$ there are two possible reasons for this. It could be that i was enumerated into A at stage s,

in which case any extension of ν_0 is permissible. Otherwise, there must be a B-change so that $\tau \not\subseteq B_{s-1}$ but $\tau \subseteq B_s$. Now we can argue as in the previous case. We must have i enumerated into A at stage $s' > |\tau|$, and since ν_0 is of length $> i$, any $\alpha \subset A_{s-1}$ such that $\nu_0 \subseteq \Phi(\alpha)$ at the end of stage $s-1$ is sufficiently long that $\tau' \subseteq \Gamma(\alpha)$, where τ' is the initial segment of B_{s-1} of length s'. Again, we conclude that any extension of ν_0 is permissible.

3. Neither of the two previous case hold, and at step k of stage s we find σ of lowest priority such that $\mu_s^* \in T_\sigma$, but is not a T_σ-leaf, and set μ_s^* to be the left successor of its previous value in T_σ. In this case let $\mu = \mu_s^*$ before its redefinition in step k. μ was a leaf of T_σ prior to stage s. The two successors of μ in T_σ were enumerated into this tree at stage s, and must both be permissible. Let μ' be the longest string which is an initial segment of both successors of μ in T_σ, and also of μ_{s-1}. We now have two cases to consider. If $\mu' \subset \mu_s$, then μ_s is permissible. Otherwise, there must be a demand (τ, i, ν_0, ν_1) such that $\nu_0 \subset \mu'$, implemented at step $k+1$ of stage s. If this demand was also implemented at step $k+1$ of stage $s-1$ then the two processes have not diverged in a strong sense. We can say that the two iterations did not *strongly diverge* at step k, since the same demand was implemented at the next step. So we choose the next step k' at which the two iterations strongly diverge. Then we have a demand that was implemented at step $k'+1$ of stage s, which was not implemented at step $k'+1$ of stage $s-1$, so we reduce to the two previous cases.

□

Lemma 5. *For all n, there exists a leftmost node of length n which is visited infinitely often. Call it σ_n. This node satisfies the following:*

1. *it is initialised only finitely many times.*

2. *if it is a $\mathcal{P}_e$ node then it ensures $\mathcal{P}_e$ is satisfied. There is some stage s such that for all stages $s' > s$, whenever σ_n receives control it plays outcome f, or there is some least m such that σ_n plays outcome m.*

3. *if it is a $\mathcal{M}_e$ node then it ensures the e-th minimality requirement is satisfied. If σ_n plays outcome ∞ in infinitely many stages, then T_{σ_n} is infinite and $M \leq_T \Psi_e(M)$. Otherwise T_{σ_n} is finite and $\Psi_e(M)$ is partial or computable.*

Proof. We proceed by induction on n. Since λ is the $\mathcal{M}_0$ node, and we assumed that Ψ_0 is the identity functional, the result for $n = 0$ is clear. So suppose $n > 0$ and that the result holds for all $n' < n$. As $\mathcal{M}$ nodes have finitely many outcomes, we look at the $\mathcal{P}$ nodes of length n'. (2) of the induction hypothesis states that all these $\mathcal{P}$ nodes eventually play outcome f or $m \in \mathbb{N}$, so each $\mathcal{P}$ node of length n' effectively has finitely many outcomes, so there is a unique σ_n of length n. There are only finitely many stages where nodes strictly to the left of σ_n are visited (otherwise, they would be σ_n) so σ_n is initialised only finitely many times in that case. Point (3) of the induction hypothesis implies that for τ that are $\mathcal{M}$ nodes with $\tau * f \subset \sigma_n$, T_τ is finite. So all $\mathcal{M}$ nodes to the left of σ_n are only visited finitely many times, and so can only enumerate finitely many splittings into their lists. Now consider the modules N^j_τ where $\tau <_L \sigma$, or $\tau * i \subseteq \sigma$ for $j < i$. They can only enumerate finitely many demands. Consider a demand (τ, j, ν_0, ν_1) issued by $N^j_{\tau_0}$. If $\tau \not\subset B$,$j \notin a$ or $\nu_0 \not\subset M$ then eventually this demand is never implemented. On the other hand, if there is a stage s where $\tau \subseteq B_s$,$k \in A_s$, and $\nu_0 \subseteq \mu_s$ then the only way in which the demand could fail to be implemented would be the implementation of a demand of higher priority (τ', k, ν_2, ν_3), such that $\nu_2 \subset \nu_0$ and $\nu_3 \supset \nu_0$. When two distinct trees T_σ and $T_{\sigma'}$ are not nested, initialisation means that all the strings in one of

the trees are of strictly greater length than all strings in the other, as we choose z_σ large after initialisation. Let σ' be the node that issues the demand (τ', k, ν_2, ν_3). Since $\nu_2 \subset \nu_0$ and $\nu_3 \supset \nu_0$ it must be the case that $T_{\tau_0^*}$ and $T_{\sigma'^*}$ are nested. Since σ' is of higher priority, $T_{\tau_0^*}$ must be equal to $T_{\sigma'^*}$, or built as a subtree of it. This contradicts the condition $\nu_2 \subset \nu_0$ and $\nu_3 \supset \nu_0$, given that ν_3 is a success or ν_2 in $T_{\sigma'^*}$. So in this list of demands there either a stage after which they are always or never implemented. Thus σ_n is only initialised finitely many times.

Now suppose σ_n is a $\mathcal{P}_e$ node. We wish to show that any demand (τ, i, ν_0, ν_1) issued by σ_n subsequent to its finial initialisation is *met*. That is, if $\tau \subset B, i \in A$ and $\nu_0 \subset M$, then $\nu_1 \subset M$. i.e., there is a stage beyond which the demand is always implemented and never injured. The argument above shows that σ_n only satisfies the conditions for initialisation at finitely many stages, and suffices to show that the demand will be implemented. Otherwise, if the demand were injured, there would need to be another demand (τ', j, ν_2, ν_3) of higher priority, at a later step of the iteration of phase 2 of that stage, where $\nu_0 \subset \nu_2 \subset \nu_1$. Initialisation means that the node τ that issued this demand can not be to the left of σ_n, since we choose z_{σ_n} large. So we must have $\tau * k \subset \sigma_n$ for some $k < j$. As ν_1 is a finite string, there are only finitely many possible value of j, and for one to be implemented, the demand must be issued prior to the stage where j is enumerated into A. Thus there are only finitely many demands of the correct form to cause injury to our demand. Since the injuring demand (τ', j, ν_2, ν_3) is issued by N_τ^j, and $\tau * k \subset \sigma_n$, with $k \leq j$, there is some stage beyond which $\tau' \not\subset B_s$, as otherwise τ would diagonalise and play outcome f. So beyond this stage the demand is not implemented, and our demand is not injured.

Now, if σ_n has outcome f eventually, or there is a least m such that σ_n eventually

always plays outcome m, then we see that $\mathcal{P}_e$ is satisfied. So, assume towards a contradiction that this does not hold, and we will show that B computes A. Then $\Psi_e(B) = M$. For each $i \notin A$, there exists $\tau \subset B$ enumerated into $\Psi_{\sigma_n}(i)$. If $i \in A$, then for any $\tau \subset B$ enumerated into $\Psi_{\sigma_n}(i)$ there is a demand (τ, i, ν_0, ν_1) issued such that $\nu_0 \subset \Psi_e(\tau)$ and ν_1 is compatible with $\Psi_e(\tau)$. Since $\Psi_e(B) = M$ we have $\nu_0 \subset M$, and there must be a stage after which this demand is always implemented, and not injured, giving the required contradiction. So (2) of the induction holds

Finally, suppose σ_n is assigned the requirement $\mathcal{M}_e$. We show that subsequent to the last initialisation of σ_n, once T_{σ_n} is non-empty, μ_s extends a leaf of T_{σ_n} at every stage for which σ_n is visited. Once this is shown, standard arguments involving the construction of minimal degrees give (3) of the induction. Let s_0 be the first stage where σ_n is visited subsequent to its last initialisation. Let $s_1 > s_0$ be the stage where we define $T_{\sigma_n}(\lambda)$. Then at every stage $s \geq s_1$ where σ_n is visited, $T_{\sigma_n}(\lambda) \subseteq \mu_s$, and all demands that are implemented were either implemented at, or before, stage s_1, or are of the form (τ, i, ν_0, ν_1) were ν_0, ν_1 in T_{σ_n}. Then μ_s extends a leaf of T_σ, as required, satisfying point (3) of the induction, and the lemma is proven. □

By the two lemmas, M exists below A, is minimal, and is not below B, so the theorem is proven.

Chapter 5

A High$_2$ Degree That Does not Satisfy the Meet Property

Theorem 7. *There exists a High$_2$ degree that does not satisfy the meet property.*

We construct A, B such that $B <_T A$, A is high$_2$, and for all $0 <_T C <_T A$ there exists $0 <_T D <_T B, C$. This suffices to show that the meet of B, C is not $\mathbf{0}$.

We now insist on the restriction that strings that e-split are of equal length. This is in reality no restriction, for if α_i is shorter then $\alpha_i * 0^k$ will still form an e-splitting with $\alpha_{\bar{i}}$.

We construct $A = \lim_{s\to\infty} \alpha_s$, $B = \lim_{s\to\infty} \beta_s$ and $D_e = \lim_{s\to\infty} \delta_{e,s}$, for all $e \in \mathbb{N}$. The aim is that if $\Psi_e(A)$ is total and incomputable then D_e is incomputable and below B and $\Psi_e(A)$.

We enumerate axioms for Turing functionals $\Phi, \{\Theta_e, \Xi_e : e \in \mathbb{N}\}$ with the aim that $\Phi(A) = B$, $\Xi_e(B) = D_e$, $\Theta_e(\Psi_e(A)) = D_e$. With regard to the previous paragraph this will only occur for e where $\Psi_e(A)$ is total and incomputable.

We let $X_e = \{p_{e+1}^n : n \in \mathbb{N}\}$, for each $e \in \mathbb{N}$, be infinite recursive sets, where p_e

is the e-th prime. We then have the requirement that A be high$_2$:

$$\mathcal{P}_e : e \notin \text{Tot}(0') \iff \exists z \forall x > z[x \in X_e \to x \notin A]$$

Satisfying all $\mathcal{P}_e$ gives A to be high$_2$ as a Σ_2 question relative to A solves membership of 0^3, so $A^2 \geq 0^3$, and the reverse inequality is automatic.

There is no way to solve this requirement all at once, so we split each $\mathcal{P}_e$ into an infinite collection $\mathcal{P}_{e,l}$

$$\mathcal{P}_{e,l} : \Psi_e(0') \upharpoonright l \downarrow \iff |X_e \cap A| > l$$

Then if $\mathcal{P}_{e,l}$ is satisfied for all l, then $X_e \cap A$ is infinite, and P_e is satisfied. If, on the other hand, $\mathcal{P}_{e,l}$ is not satisfied for some least l, then for all $l' > l$ $\mathcal{P}_{e,l'}$ is not satisfied, and $X_e \cap A$ is finite.

We also need to establish that B is actually a witness to A not having the meet property. This becomes:

$$\mathcal{Q}_e : \text{If } \Psi_e(A) \text{ total and incomputable then } \exists D_e >_T 0, D_e <_T \Psi_e(A), B$$

We split the $\mathcal{Q}_e$ requirement across nodes $\{\mathcal{Q}_{e,n} : n \in \mathbb{N}\}$ and $\{\mathcal{R}_d : d \in \mathbb{N}\}$. A $\mathcal{Q}_{e,n}$ node searches for a piece of evidence that $\Psi_e(A)$ is total and incomputable, and if it finds such evidence it passes control to a $\mathcal{R}_d$ module that attempts to diagonalise D_e against Ψ_d. $\mathcal{Q}_{e,n}$ modules attempt to show that $\Psi_e(A)$ is total and incomputable by looking for an appropriate, possibly nested, e-splitting.

The result would be trivial if $B \equiv_T A$ so we have one further requirement:

$$S_e : A \neq \Psi_e(B)$$

This is the easiest requirement to satisfy, as it only requires a standard diagonalisation like we saw in the Friedberg-Muchnik proof.

5.1 Isolated Modules

Before looking at how modules interact with each other, we first consider them in isolation. As $\mathcal{Q}, \mathcal{R}$ modules are both working to satisfy a single requirement we have their interactions in this section.

5.1.1 $\mathcal{P}_e$ Modules

Each $\mathcal{P}_e$ requirement is split across nodes $\mathcal{P}_{e,0}, \mathcal{P}_{e,1}, \cdots, \mathcal{P}_{e,l}, \cdots$, though potentially this sequence of nodes terminates. Node $\mathcal{P}_{e,l}$ waits for a stage s where $\Psi_e(0') \upharpoonright l[s] \downarrow$. If this happens then this node enumerates x from X_e into A, and plays outcome 1, where there will be a node $\mathcal{P}_{e,l+1}$. If this does not happen then the node plays outcome 0, where there will not be a $\mathcal{P}_e$ node.

If, at some later stage the $\mathcal{P}_{e,l}$ node observes that $\Psi_e(0') \upharpoonright l \uparrow$ then the node removes x from A and plays outcome 0 again, where there still will not be a $\mathcal{P}_{e,l+1}$ module.

We will let the True Path be the leftmost path of nodes visited infinitely many times. This is because if a $\mathcal{P}_{e,l}$ module plays outcomes 0 and 1 in infinitely many stages, then it observes a value of $\Psi_e(0')$ where the use goes to infinity - i.e. $\Psi_e(0') \uparrow$ and we actually want outcome 0 in this case. If for all $l \in \mathbb{N}$ there exists a $\mathcal{P}_{e,l}$ module that plays outcome 0 in finitely many stages then each module sees $\Psi_e(0') \upharpoonright l$ converge, and enumerates $x \in X_e$ into A. So $A \cap X_e$ is infinite, and $e \in \text{Tot}(0')$, as required. If instead there is some $\mathcal{P}_{e,l}$ module that plays outcome 0 infinitely many times, then there is some $n < l$ for which $\Psi_e(0'; n) \uparrow$, but this module puts a block on modules of lower priority enumerating $x \in X_e$ into A, so $A \cap X_e$ is finite, and $e \notin \text{Tot}(0')$.

5.1.2 $\mathcal{Q}_{e,n}$ Modules

This module attempts to find evidence as to whether $\Psi_e(A)$ is total and incomputable or not. It reserves a space on A and attempts to find an e-splitting above this location. If one is found then outcome 1 is played which leads to a $\mathcal{R}_d$ module. If one is not found ever then $\Psi_e(A)$ is computable if total, and the module plays outcome 0 which does not lead to a $\mathcal{R}_d$ module. If for all n the $\mathcal{Q}_{e,n}$ module finds an e-splitting then we have correctly identified $\Psi_e(A)$ as total and incomputable.

5.1.3 $\mathcal{R}_d$ Modules

A $\mathcal{R}_d$ module has been given evidence that $\Psi_e(A)$ is total in the form of an e-splitting $\alpha_0 <_L \alpha_1$. The module reserves a location on D_e, call it x, initially set to zero. If at some stage the module observes $\Psi_e(\lambda; x) \downarrow = 0 = D_e(x)$ then the module changes $D_e(x)$ to 1. The module plays outcome equal to $D_e(x)$.

When playing outcome 0 the module has $\alpha_0 \subset A$, $\beta_s \subset B$ and writes axioms $\Theta_e(\Psi_e(\alpha_0)) = D_e \upharpoonright x$, $\Xi_e(\beta_s) = D_e \upharpoonright x$. If the module moves to outcome 1 then the module changes $\alpha_1 \subset A$, and flips the last bit of β_s from 0 to 1, and writes axioms $\Theta_e(\Psi_e(\alpha_0)) = D_e \upharpoonright x$, $\Xi_e(\beta_s) = D_e \upharpoonright x$, which will be consistent with the previous axioms written as the input strings have changed.

5.1.4 $\mathcal{S}_e$ Modules

A $\mathcal{S}_e$ module reserves a location x on A and sets it to zero. If the module observes that $\Psi_e(B; x) \downarrow = 0 = A(x)$ then the module changes $A(x)$ to 1. The outcome the module plays is equal to $A(x)$. In outcome 0 axioms are written so that $\Phi(A) = B$, which are rewritten if we change $A(x)$.

$\mathcal{R}, \mathcal{S}$ modules are performing entirely standard diagonalisations, with the extra

condition that they write axioms that are consistent. This extra condition is handled in the $\mathcal{S}$ modules by encoding the outcome the module plays into A, and in the $\mathcal{R}$ modules by choosing which member of a e-splitting will be a substring of A, thus changing $\Psi_e(A)$.

5.2 Interactions Between Modules

Any module that is not a $\mathcal{P}_e$ module will not be allowed to enumerate any member of $\{x \in X_e : e \in E\}$ into A, where E is the collection of indices for $\mathcal{P}_e$ modules of higher priority. Then when a $\mathcal{Q}$ module searches for an e-splitting above α and fails we can declare the complement of $\{X_e : e \in E\}$ above α to be a partial computable e non-splitting tree.

There is work to be done in ensuring that the axioms written are consistent. We do this by having modules encode their outcomes into A, B. When a $\mathcal{P}$ module reserves a space on A, it reserves a value, enough to write 0,1, opposite to outcome it is playing. On B it reserves one bit, which will be 0 for outcome 0, but 1 for outcome 1 .

A $\mathcal{R}_e$ module does not code in A directly, but in $\Psi_e(A)$. It has been handed a pair of strings α_0, α_1 that are e-splitting. When $\mathcal{R}$ plays outcome 0, $\alpha_0 \subset A$ and the reserved spaces on B, D_e are 0. When it plays outcome 1 it ensures $\alpha_1 \subset A$ and the spaces reserved on B, D_e are 1.

An $\mathcal{S}$ module reserves a single space on A initially set to be 0. If $\mathcal{S}$ changes to outcome 1 then this is changed to 1.

$\mathcal{Q}$ is the interesting case for consistent axioms as the only restriction we can place on the search for an e-splitting is to avoid X_e of higher priority. We can not insist that $A(x) = 0$ in outcome 0 and $A(x) = 1$ in outcome 1, as if we only search

above $A(x) = 1$ and declare it to be the start of a nonsplitting tree then we have not determined that $A(x) = 0$ is the start of a nonsplitting tree. Instead we have $\mathcal{Q}$ look at existing axioms, and declare that if $\mathcal{Q}$ wishes to use a string that is a substring of some axiom, then it must use the full string associated with the axiom. So say that $\mathcal{Q}$ wishes to use α, which is extended by α^* which already lies in an axiom, and it has observed $\Psi_e(\alpha)\downarrow$. The next issue is that $\mathcal{Q}$ has not observed $\Psi_e(\alpha^*)\downarrow$. In fact, there is no guarantee that $\Psi_e(\alpha^*)\downarrow$. So this $\mathcal{Q}$ node can only declare that it has observed $\Psi_e(\alpha)\downarrow$, but it requires $\alpha^* \subset A$. A successive $\mathcal{Q}$ node will act, and will search for an e-splitting extending (an extension of) α^*. So this successive node will observe whether or not $\Psi_e(\alpha^*)\downarrow$. To ensure consistency of the axioms we therefore see that $\mathcal{Q}$ nodes must work in concert, similarly to how the $\mathcal{P}_e$ requirement is split over countably infinitely many nodes.

5.3 The Tree of Strategies

We work on a tree of strategies consisting of $\mathcal{P}, \mathcal{Q}, \mathcal{R}$ and $\mathcal{S}$ modules. The consistency requirement does not lie on the tree of strategies, but is considered to be the requirement of highest priority - it may injure any requirement and may not be injured itself. All modules each have 2 outcomes $0 <_L 1$. Every $\mathcal{P}$ module has 3 $\mathcal{Q}$ modules as immediate successors, every $\mathcal{Q}$ module has a $\mathcal{S}$ module at the end of its 0 outcome, and a $\mathcal{R}$ module at the end of its 1 outcome, every $\mathcal{R}$ module has two $\mathcal{S}$ modules as immediate successors, and every $\mathcal{S}$ module has two $\mathcal{P}$ immediate successors.

At the start of stage s control is passed to the base node λ and continues until some node halts the stage. We let the control path at stage s be the set of nodes that receive control at stage s. We let the true path be the leftmost set of nodes

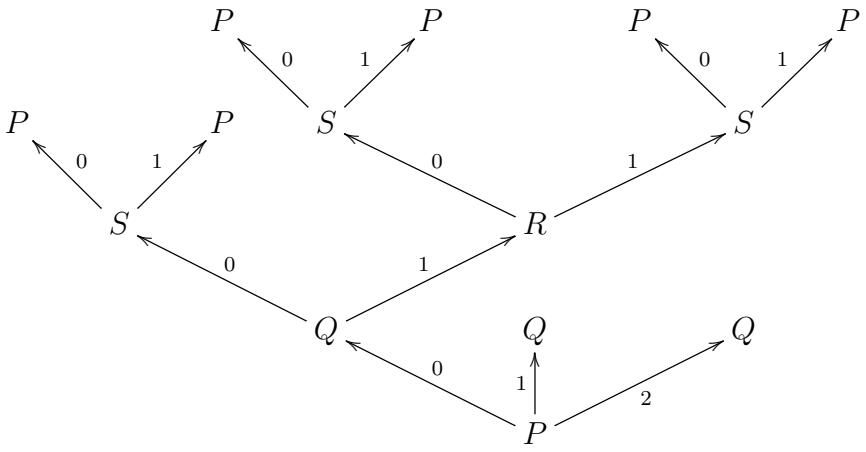

Figure 5.1: Part of the tree of strategies

that receive control in infinitely many stages.

Subscripts used to describe modules earlier will be defined the first time that a module receives control. This is because if a $\mathcal{P}_{e,l}$ module plays outcome 0 then there is no requirement for there to be another $\mathcal{P}_{e,l'}$ module on the control path, if it plays outcome 1 then we eventually require there to be a $\mathcal{P}_{e,l+1}$ module. We can computably declare in advance what index any given node on the tree of strategies will obtain, if it is given control, but it is easier to define it when it first receives control.

5.4 Construction

At stage 0 $\alpha_0, \beta_0, \delta_{e,0}$ are all defined to be λ. In any stage $s+1$ where α, β or δ_e are not redefined then we set $\gamma_{s+1} = \gamma_s$, where γ is α, β or δ_e.

At the start of stage $s > 0$ control is passed to the base node λ and continues until some node halts the stage.

We must now define what it means for a module to be *i*nitialised. Any module that is initialised forgets any values that it has stored. Any module that is initialised will initialise its immediate successors, if these modules have any stored values.

5.4.1 First Action of a Module σ in Stage s

We split by case according to the type of module σ is.

First Action of a $\mathcal{P}$ Module

σ takes the least $\langle e, l\rangle$ such that for all $\mathcal{P}$ modules $\tau \subset \sigma$ the following hold: if τ is a $\mathcal{P}_{e,m}$ module for some $m \neq l$ then $\tau * 1 \subset \sigma$. σ is then a $\mathcal{P}_{e,l}$ module.

σ stores the least $x > |\alpha_{s-1}|$ where $x \in X_e$ as a. σ then sets:

$$\alpha_s(x) = \begin{cases} \alpha_{s-1}(x) & \text{if } x \leq |\alpha_{s-1}| \\ 0 & \text{if } |\alpha_{s-1}| < x \leq a \\ \uparrow & \text{otherwise} \end{cases}$$

$$\beta_s = \beta_{s-1} * 0$$

σ writes axioms for $\Phi(\alpha_s) = \beta_s$. σ then halts the stage.

First Action of a $\mathcal{Q}$ Module

σ takes the least $\langle e, n\rangle$ such that the following hold:

- If $\tau \subset \sigma$ is a $\mathcal{Q}_{f,m}$ module then $(e, n) \neq (f, m)$.
- If $\tau \subset \sigma$ is a $Q_{e,n'}$ module for $n' \neq n$ then $\tau * 1 \subset \sigma$.

Let E be the set of values of e' such that $\tau \subset \sigma$ is a $\mathcal{P}_{e',l}$ node, for some l. Let a be the least integer greater than $|\alpha_{s-1}|$ that is not in any $\{X_e : e \in E\}$. Let $b = |\beta_{s-1}| + 1$. σ sets:

$$\alpha_s(x) = \begin{cases} \alpha_{s-1}(x) & \text{if } x \leq |\alpha_{s-1}| \\ 0 & \text{if } |\alpha_{s-1}| < x \leq a \\ \uparrow & \text{otherwise} \end{cases}$$

First Action of a $\mathcal{R}$ Module

σ takes the least f such that for all $\mathcal{R}$ modules $\tau \subset \sigma$, τ is not a $\mathcal{R}_f$ module. Then σ is a $\mathcal{R}_f$ module.

σ has been passed two triples which we may call $(\alpha_0, \alpha_0^*, \beta_0)$,$(\alpha_1, \alpha_1^*, \beta_1)$. σ also asks σ^- for its index e and stores that it is working with D_e. σ sets $\alpha_s = \alpha_0^*$ and $\beta_s = \beta_0$. σ stores $d = |\delta_{e,s-1}| + 1$ and sets $\delta_{e,s} = \delta_{e,s-1} * 0$. σ writes axioms:

$$\Phi(\alpha_0^*) = \beta_0$$

$$\Theta_e(\Psi_e(\alpha_0)) = \delta_{e,s}$$

$$\Xi_e(\beta_0) = \delta_{e,s}$$

σ then halts the stage.

First Action of a $\mathcal{S}$ Module

σ takes the least f such that for all $\mathcal{S}$ modules $\tau \subset \sigma$, τ is not an $\mathcal{S}_f$ module. Then σ is a $\mathcal{S}_f$ module.

Let E be the collection of e such that $\tau \subset \sigma$ is a $\mathcal{P}_e$ module. Let a be the least integer greater than $|\alpha_{s-1}|$ and $|\beta_{s-1}|$ not in $\{X_e : e \in E\}$.

σ sets:

$$\alpha_s(x) = \begin{cases} \alpha_{s-1}(x) & \text{if } x \leq |\alpha_{s-1}| \\ 0 & \text{if } |\alpha_{s-1}| < x \leq a \\ \uparrow & \text{otherwise} \end{cases}$$

$$\beta_s = \begin{cases} \beta_{s-1}(x) & \text{if } x \leq |\beta_{s-1}| \\ 0 & \text{if } |\beta_{s-1}| < x \leq a \\ \uparrow & \text{otherwise} \end{cases}$$

σ then writes axioms for $\Phi(\alpha_s) = \beta_s$. σ then halts the stage.

5.4.2 Subsequent Actions of a Module σ in Stage s

Again, we split by case according to what type of module σ is.

Subsequent Action of a $\mathcal{P}$ Module

σ tests if

$$\Psi_e(0') \upharpoonright l[s] \downarrow$$

If so, and σ last played outcome 1, then σ plays outcome 1. If not, and σ last played outcome 0, then σ plays outcome 0.

If so, and σ last played outcome 0 then σ sets:

$$\alpha_s(x) = \begin{cases} \alpha_{s-1}(x) & \text{if } x < a \\ 1 & \text{if } x = a \\ 0 & \text{if } x = a+1 \\ \uparrow & \text{otherwise} \end{cases}$$

$$\beta_s(x) = \begin{cases} \beta_{s-1}(x) & \text{if } x < b \\ 1 & \text{if } x = b \\ \uparrow & \text{otherwise} \end{cases}$$

Then σ plays outcome 1. If not, then σ plays outcome 0.

If σ has played outcome 1 but not outcome 2 then σ tests if

$$\Psi_e(0') \upharpoonright l[s] \uparrow$$

If so then σ sets

$$\alpha_s(x) = \begin{cases} \alpha_{s-1}(x) & \text{if } x < a \\ 0 & \text{if } x = a \\ 1 & \text{if } x = a+1 \\ \uparrow & \text{otherwise} \end{cases}$$

and plays outcome 2. If not, then σ plays outcome 1.

If σ has ever played outcome 2, then σ plays outcome 2.

Subsequent Action of a $\mathcal{Q}$ Module

If σ has ever played outcome 1 then it plays outcome 1. Otherwise, σ spends s stages searching for an e-splitting above $\alpha_s \upharpoonright a$ avoiding $\{X_e : e \in E\}$. If none is found then outcome 0 is played. Otherwise, we let the e-splitting be $\alpha_0 <_L \alpha_1$.

If there exist axioms of the form $\Phi(\alpha_i') = \beta_i'$ for some $\alpha_i' \supseteq \alpha_i$ then let α_i^* be the greatest α_i', and let its axiom be $\Phi(\alpha_i^*) = \beta_i^*$. This may be the case for both, neither, or only one $i \in \{0, 1\}$. If α_i^* does not exist then it is set to α_i. If β_i^* does not exist then it is set to $\beta_{s-1} * i$. If α_i^* is shorter than $\alpha_{\bar{i}}^*$ then pad the shorter string with zeroes until they are of the same length. If β_i^* is shorter than $\beta_{\bar{i}}^*$ then extend it to be incompatible with $\beta_{\bar{i}}^*$ but of the same length.

σ then passes a pair of triples to $\sigma * 1$: $(\alpha_0, \alpha_0^*, \beta_0^*)$,$(\alpha_1, \alpha_1^*, \beta_1^*)$ and plays outcome 1.

Subsequent Action of a $\mathcal{R}$ Module

If σ has ever played outcome 1, then σ plays outcome 1. Otherwise σ tests if:

$$\Psi_f(\emptyset; d)[s] \downarrow = 0$$

If so then σ sets:

$$\alpha_s(x) = \alpha_1^*$$

$$\beta_s(x) = \beta_1$$

$$\delta_{e,s}(x) = \begin{cases} \delta_{e,s-1}(x) & \text{if } x \neq d \\ 1 & \text{if } x = d \end{cases}$$

Then σ writes axioms for:

$$\Xi_e(\beta_s) = \delta_{e,s}$$

$$\Theta_e(\Psi_e(\alpha_1)) = \delta_{e,s}$$

Finally σ plays outcome 1. Otherwise σ plays outcome 0.

Subsequent Action of a $\mathcal{S}$ Module

If σ has ever played outcome 1, then σ plays outcome 1. Otherwise σ tests if:

$$\Psi_e(\beta_{s-1}; a)[s] \downarrow = 0$$

If so then σ sets

$$\alpha_s(x) = \begin{cases} \alpha_{s-1}(x) & \text{if } x < a \\ 1 & \text{if } x = a \\ \uparrow & \text{otherwise} \end{cases}$$

$$\beta_s = \beta_{s-1}$$

σ then writes axioms for $\Phi(\alpha_s) = \beta_s$ and plays outcome 1. Otherwise σ plays outcome 0.

5.5 Verification

As the stage halts when a node acts for the first time, the control path in stage s can consist of at most $s + 1$ nodes. We let the True Path be the leftmost path of nodes that receive control in infinitely many stages. Every search that a node performs is bounded, and only uses already computed information. So every node can perform its instructions, and every stage halts. Every $\mathcal{P}$ node extends α, β, so $A = \lim_{s\to\infty} \alpha_s, B = \lim_{s\to\infty} \beta_s$ are well-defined. If there are infinitely many suitable $\mathcal{R}$ nodes on the true path then each one extends δ_e, so $D_e = \lim_{s\to\infty} \delta_{e,s}$ is well-defined.

To verify that the requirement $\mathcal{P}_e$ is satisfied we first observe that every node of lower priority than a $\mathcal{P}_e$ node avoids X_e, so only finitely many nodes do not avoid X_e, each of which can enumerate finitely many elements of X_e into A. So, provided that the $\mathcal{P}_e$ modules are performing properly, the requirement $\mathcal{P}_e$ is satisfied.

For fixed e, if there is some node σ that is a $\mathcal{P}_{e,l}$ node on the True Path such that $\sigma * 0$ is on the True Path, then for every stage s where $\Psi_e(0') \upharpoonright l[s] \downarrow$, there is some $s' > s$ where the computation diverges. Therefore $\Psi_e(0') \upharpoonright l \uparrow$, and $A \cap X_e$ is finite.

Otherwise, there are infinitely many $\mathcal{P}_e$ nodes that eventually only play outcome 1 on the True Path. Each of these nodes has a stage s such that for all $s' > s$ it observes $\Psi_e(0') \upharpoonright l[s'] \downarrow$, from which we conclude that $\Psi_e(0') \upharpoonright l \downarrow$, and we require $X_e \cap A$ to be infinite. Each of these nodes enumerates $x \in X_e$ into A and never removes it, so $X_e \cap A$ is infinite and the requirement is satisfied.

To see that requirement $\mathcal{Q}_e$ is satisfied, we see whether there are infinitely many $\mathcal{Q}_e$ modules on the true path or not. If not then there is some $\mathcal{Q}_e$ module σ that always plays outcome 0. In this case $\mathcal{Q}_e$ does not find an e-splitting above $\alpha_s \upharpoonright a$,

avoiding $\{X_e : e \in E\}$, so there must not be such an e-splitting. Therefore the complete tree above $A \upharpoonright a$ of strings that avoid $\{X_e : e \in E\}$ is a p.c. nonsplitting tree that A lies on. Therefore $\Psi_e(A)$ is computable, if total, and the requirement is satisfied.

If otherwise there are infinitely many $\mathcal{Q}_e$ modules on the true path, then each one finds an e-splitting, possibly extends it to make axioms consistent, and passes it to an $\mathcal{R}$ module. Each e-splitting extends a previous e-splitting, so $\Psi_e(A)$ must be total. $\Psi_e(A)$ bounds D_e, which we will see is incomputable, so $\Psi_e(A)$ must be incomputable.

If there are infinitely many $\mathcal{Q}_e$ modules on the true path, then for each f there is a $\mathcal{R}_f$ module working immediately above $\mathcal{Q}_e$. If $\mathcal{R}_f$ plays outcome 1 on the true path, then at some stage it observes $\Psi_f(\emptyset)(x) \downarrow = 0$, but it sets $D_e(x) = 1$, so D_e does not equal $\Psi_f(\emptyset)$. If $\mathcal{R}_f$ plays outcome 0 on the true path then $\Psi_f(\emptyset)(x) \uparrow$ or $\downarrow = 1$. But in this case $D_e(x) = 0$ so $D_e \neq \Psi_f(\emptyset)$. In either case we have D_e incomputable, as required.

The $\mathcal{S}_e$ module on the true path does a similar diagonalisation - if it observes some stage where $\Psi_e(B; x) \downarrow = 0$ then it changes $A(x)$ to 1, otherwise it knows that $A(x) = 0$ is distinct from $\Psi_e(B; x)$. So $\Psi_e(B) \neq A$, for all e, and A is not computable from B.

To establish consistency of the axioms for $\Phi(A) = B$ assume, towards a contradiction, that nodes σ, τ write contradictory axioms. We now split by 3 cases: $\sigma = \tau, \sigma \subset \tau, \sigma | \tau$. If $\sigma = \tau$ then we split by case according to what type of node it is. It can not be a $\mathcal{Q}$ as these never write axioms to Φ. If it is a $\mathcal{P}$ module then one axiom must be of the form $\Phi(\alpha * 0) = \beta_0$, and the other $\Phi(\alpha * 1) = \beta_1$, where α, β are some strings. But these axioms have incompatible strings as input, so are

consistent. If it is a $\mathcal{R}$ module then one axiom is $\Phi(\alpha_0^*) = \beta_0$ and the other axiom is $\Phi(\alpha_1^*) = \beta_1$. But σ^- established that substrings of α_0^*, α_1^* form an e-splitting, so α_0^*, α_1^* must be incompatible, and these axioms are consistent. If σ is a $\mathcal{S}$ module then one axiom is of the form $\Phi(\alpha * 0) = \beta_0$ and one is of the form $\Phi(\alpha * 1) = \beta_1$, so the input strings are incompatible and the axioms are consistent. So $\sigma \neq \tau$.

Without loss of generality, let σ act before τ. Now consider the case $\sigma \subset \tau$. By the above reasoning we may consider the last axiom that σ wrote before τ writes axioms. If σ writes an axiom of the form $\Phi(\alpha) = \beta$ then τ writes an axiom of the form $\Phi(\alpha * \alpha') = \beta'$, so we only require $\beta \subseteq \beta'$ for consistency of axioms. But the location τ works on B must be greater than the location σ works on B, so $\beta \subset \beta'$. So if $\sigma \subset \tau$ then the axioms are consistent. So we move on to the next case.

Now consider the case $\sigma|\tau$. Let $\rho = \sigma \wedge \tau$. By earlier reasoning σ writes axioms consistent with ρ, which changes outcome, and in changing outcome, keeps consistent axioms, and ρ writes consistent axioms with τ. So, therefore, σ writes consistent axioms with τ.

So we conclude that $\sigma \neq \tau, \sigma \not\subseteq \tau$, $\tau \not\supseteq \sigma$ and $\sigma \not| \tau$, giving our contradiction, so the axioms for Φ are consistent.

We now verify consistency of axioms for $\Theta_e(\Psi_e(A)) = D_e$ and $\Xi_e(B) = D_e$. Again, towards a contradiction, let σ,τ write inconsistent axioms for either of these functionals. If $\sigma = \tau$ then one pair of axioms is of the form $\Theta_e(\Psi_e(\alpha_0)) = \delta_{e,s}$ and $\Xi_e(\beta_0) = \delta_{e,s}$ and the other pair is of the form $\Theta_e(\Psi_e(\alpha_1)) = \delta_{e,s'}$ and $\Xi_e(\beta_1) = \delta_{e,s'}$. α_0, α_1 form an e-splitting, so $\Psi_e(\alpha_0)|\Psi_e(\alpha_1)$, so the axioms for Θ_e are consistent. β_0, β_1 disagree on final bit, so are incompatible, so the axioms for Ξ_e are consistent. So $\sigma \neq \tau$, so let σ act before τ.

Now consider the case $\sigma \subset \tau$. Again, we may consider only the last axioms that

σ writes before τ writes axioms. In that case, every string that τ is working with is a proper extension of a string σ is working with, and the axioms are consistent. So $\sigma \not\subset \tau$.

Now consider the case $\sigma|\tau$, and let $\rho = \sigma \wedge \tau$. By earlier reasoning ρ's axioms are consistent with itself, and with σ and τ. So σ and τ have consistent axioms. So $\sigma \not| \tau$.

Therefore $\sigma \not\subseteq \not\supseteq \tau$ and $\sigma \not| \tau$, a contradiction. So our assumption that the axioms are inconsistent was false, and the axioms for Θ_e, Ξ_e are consistent.

Therefore we have built A, B and D_e such that the requirements are satisfied, and the theorem is proven.

Bibliography

[CE87] S Barry Cooper and Richard L Epstein. Complementing below recursively enumerable degrees. *Annals of Pure and Applied Logic*, 34(1):15–32, 1987.

[Coo71] Stuart Barry Cooper. *Degrees of unsolvability.* PhD thesis, University of Leicester, 1971.

[Coo74] S. Barry Cooper. Minimal pairs and high recursively enumerable degrees. *The Journal of Symbolic Logic*, 39(04):655–660, 1974.

[Coo89] S. Barry Cooper. The strong anticupping property for recursively enumerable degrees. *The Journal of Symbolic Logic*, 54(02):527–539, 1989.

[Coo03] S Barry Cooper. *Computability theory.* CRC Press, 2003.

[DLPNR16] Benedict Durrant, Andy Lewis-Pye, Keng Meng Ng, and James Riley. Computably enumerable Turing degrees and the meet property. *Proceedings of the American Mathematical Society*, 144(4):1735–1744, 2016.

[Eps75] Richard L Epstein. *Minimal degrees of unsolvability and the full approximation construction*, volume 162. American Mathematical Soc., 1975.

[Eps79] Richard Epstein. Degrees of unsolvability, structure and theory. *Lecture Notes in Mathematics*, 759, 1979.

[Eps81] Richard L Epstein. *Initial segments of degrees below 0'*, volume 241. American Mathematical Soc., 1981.

[Fri57] Richard M Friedberg. Two recursively enumerable sets of incomparable degrees of unsolvability (solution of Post's problem, 1944). *Proceedings of the National Academy of Sciences of the United States of America*, 43(2):236, 1957.

[Har82] Leo A Harrington. A gentle approach to priority arguments. *Mimeographed Notes*, 1982.

[Ish03] Shamil Ishmukhametov. On a problem of Cooper and Epstein. *The Journal of Symbolic Logic*, 68(01):52–64, 2003.

[JJ77] Carl G Jockusch Jr. Simple proofs of some theorems on high degrees of unsolvability. *Canadian Journal of Mathematics*, 29(5):1072–1080, 1977.

[Lac76] Alistair H Lachlan. A recursively enumerable degree which will not split over all lesser ones. *Annals of Mathematical Logic*, 9(4):307–365, 1976.

[LP04] Andrew Lewis-Pye. Minimal complements for degrees below 0'. *The Journal of Symbolic Logic*, 69(04):937–966, 2004.

[LP12] Andrew EM Lewis-Pye. The search for natural definability in the turing degrees. *Unpublished lecture notes*, 2012.

[LY98] Angsheng Li and Dongping Yang. Bounding minimal degrees by computably enumerable degrees. *The Journal of Symbolic Logic*, 63(04):1319–1347, 1998.

[Muc56] Albert A Muchnik. On the unsolvability of the problem of reducibility in the theory of algorithms. In *Dokl. Akad. Nauk SSSR*, volume 108, page 1, 1956.

[NSS98] André Nies, Richard A Shore, and Theodore A Slaman. Interpretability and definability in the recursively enumerable degrees. *Proceedings of the London Mathematical Society*, 77(02):241–291, 1998.

[Odi92] Piergiorgio Odifreddi. *Classical recursion theory: The theory of functions and sets of natural numbers*. Elsevier, 1992.

[Odi99] Piergiorgio Odifreddi. *Classical Recursion Theory, Vol II, volume 143 of*. 1999.

[Pos03] Emil L Post. Recursively enumerable sets of positive integers and their decision problems. *Mathematical Logic in the 20th Century*, page 352, 2003.

[Sac63] Gerald E Sacks. Recursive enumerability and the jump operator. *Transactions of the American Mathematical Society*, pages 223–239, 1963.

[Soa74] Robert I Soare. Automorphisms of the lattice of recursively enumerable sets part i: Maximal sets. *Annals of Mathematics*, pages 80–120, 1974.

[Spe56] Clifford Spector. On degrees of recursive unsolvability. *Annals of Mathematics*, pages 581–592, 1956.

[SS89] Theodore A Slaman and John R Steel. Complementation in the Turing degrees. *The Journal of Symbolic Logic*, 54(01):160–176, 1989.

[SS99] Richard A Shore and Theodore A Slaman. Defining the Turing jump. *Mathematical Research Letters*, 6(5/6):711–722, 1999.

[Tur36] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 58:345–363, 1936.

[Tur39] Alan Mathison Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, 2(1):161–228, 1939.

[Yat70] CE Mike Yates. Initial segments of the degrees of unsolvability part ii: Minimal degrees. *The Journal of Symbolic Logic*, 35(02):243–266, 1970.