The Meet Property in Local Degree Structures

Benedict Richard Fabian Durrant

Submitted in accordance with the requirements for the degree of Doctor of Philosophy

The University of Leeds

Department of Pure Mathematics

May 2014

Joint Work

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

Acknowledgements

I would like to thank my supervisors Andrew Lewis-Pye and Barry Cooper for their support and guidance during my PhD, and for their assistance in helping me secure funding. I would also like to thank my parents for their on going support and Emily for her unfailing encouragement in the final stages of my work.

Abstract

In this thesis we look at whether two different classes of local Turing degrees (the $c.e.$ degrees, and the 1-generic degrees below $\mathbf{0}'$) satisfy the meet property - where a degree $\mathbf{a}$ satisfies the meet property if it is incomputable and for all $\mathbf{b} < \mathbf{a}$ there exists a non-zero degree $\mathbf{c}$ such that $\mathbf{a} \wedge \mathbf{c} = \mathbf{0}$. We first give a general discussion of the Turing Degrees and certain known results, before giving a brief introduction to priority arguments. This is followed by some more technical considerations (full approximation and minimal degree constructions) before the proof of two new theorems - the first concerning $c.e.$ degrees and the meet property and the second concerning $1 - generic$ degrees and the meet property.

Chapter 1 contains a broad introduction to the Turing Degrees, and Chapter 2 to the Local Degrees. In Chapter 3 we consider minimal degree constructions, which we use in Chapter 4 to prove our first new theorem - Theorem 4.2.1 *Given any non-zero c.e. degree* $\mathbf{a}$ *and any degree* $\mathbf{b} < \mathbf{a}$, *there is a minimal degree* $\mathbf{m} < \mathbf{a}$ *such that* $\mathbf{m} \not\leq \mathbf{b}$. From which we get Corollary 4.2.2 *Every c.e. degree satisfies the meet property* - answering a question first asked by Cooper and Epstein in the 1980s. In Chapter 5 we prove the second new theorem - Theorem 5.2.2 *There exists a* $1 - generic$ *degree which does not satisfy the meet property* - showing that a result from Kumabe in the 1990s does not extend to the case $n = 1$.

Contents

Joint Work iii

Acknowledgements v

Abstract vii

Contents ix

List of figures xiii

1 Introduction **1**

1.1 Notation and Conventions 3

1.2 Trees 8

1.3 Priority Arguments 9

2 The Local Degrees **15**

2.1 The High-Low Hierarchy 16

2.2 Natural Definability 27

2.3 Properties in the Local Degrees 28

3 Minimal Degree Constructions **30**

3.1 Splitting Trees 31

3.2 A Minimal Degree below $\mathbf{0}''$ 34

3.2.1 Construction 35

3.2.2 Verification 35

3.3 A Minimal Degree below $\mathbf{0}'$ 36

3.3.1 Construction 38

3.3.2 Verification 39

3.3.3 Remarks 40

3.4 A Minimal Degree by Full Approximation 40

3.4.1 Construction 43

3.4.2 Verification 44

4 Computably Enumerable Degrees and the Meet Property **49**

4.1 A Minimal Degree below a C.E. Degree 50

4.1.1 Constructions below a C.E. Degree 50

4.1.2 Construction 52

4.1.3 Verification 54

4.2 C.E. Degrees and the Meet Property 56

4.2.1 Outline Proof 58

4.2.2 Construction 61

4.2.3 Verification 65

5 1-Generic Degrees and the Meet Property 72

5.1 A 1-Generic by Full Approximation 74

5.2 1-Generic Degrees and the Meet Property 76

5.2.1 Outline Proof 77

5.2.2 Tree of Strategies 88

5.2.3 Formal Construction 91

5.2.4 Verification 98

Bibliography 107

List of figures

1.1 Friedberg-Muchnik Tree of Strategies 12

3.1 A Ψ_i splitting above a string σ 31

3.2 Tree of Strategies for the Minimal Degree by Full Approximation Construction 42

3.3 Possible Outcomes on the Minimal Degree Tree of Strategies 43

4.1 Tree of Strategies for the C.E. Meet Construction 58

5.1 Example Cluster 85

Chapter 1

Introduction

This chapter gives a brief overview of Computability Theory, and includes key definitions and concepts used later in this thesis. The majority of the the content is assumed, and little motivation is given. The reader is referred to, for instance, [Coo04] or [Odi92] and [Odi99] for a more detailed technical introduction to the area.

Informally we can consider an algorithm to be an effective procedure (given some finite list of instructions) for calculating the answer to a given question - for example the calculation of the output of a given function on a certain input, or deciding whether a given element is in a specified set (e.g. the set of prime numbers). Although the informal notion of an algorithm has been around for many years (the word itself deriving from the Latin form of the name of the 9^{th} century Persian mathematician al-Khwarizmi[1]), it is only relatively recently that the concept has been formalised.

Turing's 1936 paper [Tur36], written in response to Hilbert's *Entscheidungsproblem*, gave the first widely accepted formalisation of the notion of an algorithm. Turing did this via the Turing Machine, and it is this which provides us with the modern notion of computability[2]. We consider a function on the natural numbers to be computable if it

[1]See [AD77] for more details on this.

[2]Other sufficiently strong notions of computability can be shown to be equivalent.

can be computed on a Turing Machine. The technical details of the construction of the Turing Machine are assumed, further details can be found in [Coo04].

Normally in computability we work with functions from $A \subseteq \omega$ to ω, if $A = \omega$ we call the function total, otherwise we call it partial. A characteristic function of a subset A of ω is a function $f : \omega \to \{0, 1\}$ which on input n outputs 0 if n is not in A and 1 if n is in A. We identify subsets of ω with their characteristic function.

The construction of a Turing Machine allows us to assign each a unique *Gödel Number*. By the *Padding Lemma* we have that for a given partial computable function f, there exist infinitely many i, such that i is the Gödel Number of a machine which computes f. For simplicity of numbering we assume that for any n if n is not the Gödel number of some Turing Machine, then M_n is a Turing Machine running the empty program. This allows us to list all possible Turing Machines in some effective order: $M_0, M_1, \ldots, M_n, \ldots$. We are also able, given any number n, to find the Turing Machine it represents. Given a list of all Turing Machines, we are able to pick a partial computable function $f(x, y) = M_x(y)$, which is able to represent any Turing Machine on the list. A machine which computes such a function is called a Universal Turing Machine.

Turing's paper gave the first use of a diagonalisation argument to produce an incomputable set, and thus showed that a general solution to the *Entscheidungsproblem* is impossible. This set is now known as the *halting* set. It is denoted either $\emptyset'$ or K and is defined to be, $\{x : \psi_x(x) \text{ is defined}\}$, where ψ_x is the x^{th} partial computable function.

Three years after the introduction of Turing Machines, Turing introduced, in [Tur39], the concept of an oracle Turing Machine (where a Turing Machine may ask if a certain element is contained in the given - oracle - set), and so of relative computability. Technical details of the construction of an oracle Turing Machine can again be found in [Coo04]. We write $\Psi(B) = A$ to mean that given set B, as an oracle, the Turing Machine Ψ can compute A, denoted $A \leq_T B$. We use Ψ_i to denote the *Turing Functional* which is the functional computed by the i^{th} Turing Machine - where we use functional to indicate that the machine may take both elements of 2^ω (i.e., oracle sets) as well as natural numbers as inputs. In 1944 Post [Pos44] defined the Turing degrees using this idea. If there are two Turing functionals Ψ and Ψ' (not necessarily the same) such that $\Psi(A) = B$ and $\Psi'(B) = A$ then we say that A and B are Turing equivalent, and denote this $A \equiv_T B$. The *Turing Degree* of a set $A \subset \omega$ is $deg(A) = \{X \subset \omega : X \equiv_T A\}$, usually denoted **a**. The partially ordered set of all degrees is denoted $\mathcal{D}$ and has its ordering given by $\leq_T$, namely $\mathbf{a} \leq \mathbf{b} \Leftrightarrow A \leq_T B$. The structure of $\mathcal{D}$, is a major area of study in computability and has been since its introduction.

In [KP54] Kleene and Post combined their ideas by looking at the relativisation of the halting set. They did this in the natural way - the halting set, relative to a set A, is the set $\{x : \Psi_x(A, x) \text{ is defined}\}$. This is denoted A', and is called the jump of A. The $(n + 1)^{th}$ jump of A is the jump of the n^{th} jump of A, namely $A^{n+1} = (A^n)'$. The jump of a degree **a** follows naturally: $\mathbf{a}' = \deg(A')$. They go on to define the *jump function* $f : \mathcal{D} \to \mathcal{D}$, where $f(\mathbf{a}) = \mathbf{a}'$. Within the structure of $\mathcal{D}$, specific interest is given to the *local* degrees, those below $\mathbf{0}' = \deg(\emptyset')$. Further details of this, along with results relevant to this thesis, are found in Chapter 2.

The above is a (very) brief introduction to computability theory, and a working knowledge of the material covered so far is assumed. A broader overview of the history of the area can be found in [ASF06].

1.1 Notation and Conventions

In this section we lay out the framework of notation, and conventions, that we will work within - any variations from this will be clearly explained. We include it for clarity and to remove any confusion caused by the changes in notation over the past 70 years of study.

We use lower case Roman letters from the end of the alphabet to denote natural numbers, and capital Roman letters to denote subsets of ω. We use lower case Greek letters to denote strings. Typically the strings we consider are elements of $2^{<\omega}$, though more generally a string may be from ω^{ω}. We let λ represent the empty string, and let $\sigma * \tau$ be the concatenation of (the finite strings) σ and τ. For a finite string σ we use $|\sigma|$ to denote the length of σ. We use the notation $\sigma \upharpoonright_n$ to denote the initial segment of σ of length n. τ is an extension of σ if $\tau \upharpoonright_{|\sigma|} = \sigma$. We use the notation $\sigma \subset \tau$ if τ is a proper extension of σ, and we write $\sigma \subseteq \tau$ to allow for the possibility of non-proper extension. We consider strings to be ordered by length first and then lexicographically. Given two (possibly partial) functions f and g, we write $f \subseteq g$ if the domain of f is a subset of the domain of g and for all n in the domain of f, $f(n) = g(n)$. Two functions are incompatible if, for some n, $f(n) \downarrow \neq g(n) \downarrow$, where we use the notation $f(n) \downarrow$ to mean that the function f is defined on input n. If a f is not defined on input n, we denote this $f(n) \uparrow$.

We have a standard pairing function, which is a computable bijection $\omega \times \omega \rightarrow \omega$. We use $\langle x, y \rangle$ to denote the output of our pairing function on input (x, y). If desired we are able to nest these bijections as follows $\langle \bullet, \cdots, \langle \bullet \langle \bullet, \bullet \rangle \rangle \cdots \rangle$ to get a bijection $\omega \times \cdots \times \omega \rightarrow \omega$. We let $A^{[i]}$ denote the i^{th} column of A, i.e., the set of all numbers in A of the form $\langle i, j \rangle$. Given a finite $F \subset \omega$, let $F = \{a_0 < \cdots < a_n\}$ and for each $i \leq n$ let X_i be $A^{[a_i]}$. Then we define $A^{[F]} = \bigcup_{i=0}^{n} X_i$.

As noted before we are able to list all Turing functionals in some effective list, and from now on we assume that this list is fixed. We let Ψ_i denote the i^{th} functional from it and use the convention that Ψ_0 is the identity functional - that is the functional such that $\Psi_0(A, x) = A(x)$ for any A and x. We use other upper case Greek letters to represent specific Turing functionals - either ones we construct or which compute know functions. Turing functionals compute in stages and we write $\Psi_i(A, y)[s] \downarrow = x$ to mean that the i^{th} Turing functional (from our list) with oracle A is defined on input y in at most s steps, and is equal on this input to x. If $\Psi_i(A, y)[s] \downarrow$, then for all $t \geq s$, $\Psi_i(A, y)[t] \downarrow = \Psi_i(A, y)[s]$.

We write $\Psi_i(A, y)[s] \uparrow$ to mean the functional is not defined in s steps. We write $\Psi_i(y)$ as shorthand for $\Psi_i(\emptyset, y)$, i.e., when the functional uses the empty set as an oracle.

We write $\Psi_i(A)$ to denote the (possibly) partial function which on argument n is equal to $\Psi_i(A, n)$, and in this context we may write Ψ_i as shorthand for $\Psi_i(\emptyset)$. When we are dealing with subsets of ω we identify them with their characteristic functions - so it makes sense to say, for instance, $\Psi(A) = B$.

The *use* of an oracle computation $\Psi_i(A, y)$ is $x + 1$, where x is the largest number, such that $A(x)$ is queried in the oracle computation. We write $u(A, i, y)$ for the *use* in computing $\Psi_i(A, y)$, and assume that if $\Psi_i(A, y)[s] \downarrow$ then $u(A, i, y) < s$. We may also write $u(A, i, y, s)$ for the *use* in computing $\Psi_i(A, y)[s]$. If a computation halts then it does so after a finite number of stages, and so its *use* is finite. We may write $\Psi_i(\sigma, y) \downarrow = x$ if $\Psi_i(A, y) \downarrow = x$ and $u(A, i, y) < |\sigma|$, where σ is an initial segment of (the characteristic sequence of) A. This means that if $\Psi_i(A, y) \downarrow = x$, then there exists some $\sigma \in 2^{<\omega}$, such that $\Psi_i(B, y) \downarrow = x$ for all $B \supset \sigma$. The *use* principle states the following: let $\Psi(A)$ be a converging oracle computation, and let B be a set such that $B \upharpoonright_{u(A,n)} = A \upharpoonright_{u(A,n)}$ then $\Psi(B) \upharpoonright_n = \Psi(A) \upharpoonright_n$.

Given two finite strings σ and τ we abuse notation and may say that $\Psi_i(\sigma)$ is an initial segment of $\Psi_i(\tau)$ (denoted $\Psi_i(\sigma) \subseteq \Psi_i(\tau)$) if given the greatest n such that $\Psi_i(\sigma, n) \downarrow$ there is some $n' \geq n$ such that $\Psi_i(\tau, n') \downarrow \supseteq \Psi_i(\sigma, n)$.

A set A is computably enumerable (c.e.) if there is an effective process for enumerating all the members of A. Formally we have that A is c.e. iff $A = \emptyset$ or there is a computable function f such that $A = range(f)$. All computable sets are c.e. (we simply enumerate the set in order, using the function which computes it) and if both A and $\bar{A}$ are c.e. then A is computable. Not all c.e. sets are computable, however. $\emptyset'$ is an example of such a set.

In the enumeration of a c.e. set each element may only be enumerated once, and may never be removed. We are able to expand on this by increasing the number of times an element may be entered and subsequently removed.

Definition 1.1.1 *A is $n-c.e.$ if an element $x \in \omega$ may be enumerated and removed at most n times during the enumeration of A, i.e., $A_s(x) \neq A_{s+1}(x)$ at most n times during the enumeration of A and $A_0(x) = 0$. A is $\omega - c.e.$ if there is a computable bound on the number of such enumerations and removals.*

A set A is c.e. iff it is $1-c.e.$ and is $n-c.e.$ if it can be expressed as a boolean combination of n many $c.e.$ sets, and their complements.

The following two theorems will be used throughout the thesis, proofs for them can be found in [Coo04] and [Soa87].

Theorem 1.1.2 *Normal Form Theorem*

The following are equivalent

1. A is c.e.;
2. $A \in \Sigma^0_1$;
3. $A = domain(\psi_i)$ for some i.

This means we are able to list the c.e. sets in an effective order. We use W_i to denote $domain(\Psi_i)$, and call it the i^{th} c.e. set. We use the notation $W_{i,s}$ to denote the set of numbers enumerated into W_i by the end of stage s. We use the notation W_i^A to mean the i^{th} c.e. set with respect to A.

Theorem 1.1.3 *Post's Theorem*

Given $A \subseteq \omega$ and $n \geq 0$, then

1. $\emptyset^{(n+1)}$ is Σ^0_{n+1}-complete;

2. $A \in \Sigma^0_{n+1} \Leftrightarrow A$ is c.e. in $\emptyset^{(n)}$;

3. $A \in \Delta^0_{n+1} \Leftrightarrow A \leq_T \emptyset^n$.

We are especially interested in Δ^2_0 sets, and have the following definition.

Definition 1.1.4 *A computable sequence $\{A_s\}_{s\in\omega}$ of finite sets is a Δ^0_2 approximating sequence for A if $(\forall x)lim_{s\to\infty}A_s(x)$ exists and is equal to $A(x)$. A_s is the approximation to A at stage s. For shorthand if $\{A_s\}_{s\in\omega}$ is a approximating sequence for A, we denote this $lim_s A_s = A$.*

As this thesis is focused on those sets which have approximating sequences we give a proof of the following.

Theorem 1.1.5 *The Limit Lemma*

$A \leq_T \emptyset'$ if and only if A has a Δ^0_2 approximating sequence. Similarly A is limit computable in B if and only if $A \leq_T B'$.

Proof: Let $\{\emptyset_s\}$ be a computable enumeration of the $\emptyset$ First we show that if $A \leq_T \emptyset'$ then A is Δ^0_2. $A \leq_T \emptyset'$, so we have a Turing functional Ψ_i such that $A = \Psi_i(\emptyset')$. Then $A_s = \{x < s : \Psi_i(\emptyset'_s, x)[s] = 1\}$ is the required approximating sequence for A.

Now we show that if A is Δ^0_2 then $A \leq_T \emptyset$ We construct a *change set* C, which is a c.e. set such that $A \leq_T C$ (as C is c.e., $C \leq_T \emptyset'$). We use C to record the changes in the δ^0_2 approximation to A. If $A_s(x) \neq A_{s+1}(x)$, then we enumerate $\langle x, i\rangle$ into C_{s+1}, where i is the least a such that $\langle x, a\rangle \notin C_s$. To compute A from C, on input x using the oracle C we compute the least i such that $\langle x, i\rangle \notin C$. If i is even then $A(x) = A_0(x)$, otherwise $A(x) = 1 - A_0(x)$. So $A \leq_T C \leq_T \emptyset'$.

□

Through out the thesis we will use the equivalences between being c.e. and Σ^0_1 and being Δ^0_2 and computable in $\emptyset'$. We will not cite them when doing so.

1.2 Trees

The trees we work with will normally be binary trees. The definitions given below easily generalise to non binary trees. Given $i \in \{0,1\}$, we let $\bar{i}$ denote $1-i$. We then define a *tree* to be a possibly partial function $T : 2^{<\omega} \to 2^{<\omega}$ subject to the following conditions: for all $\sigma \in 2^{<\omega}$ and $i \in \{0,1\}$, when $T(\sigma * i)\downarrow$

1. $T(\sigma)\downarrow\subset T(\sigma * i)$
2. $T(\sigma * \bar{i})\downarrow$ and is incompatible with $T(\sigma * i)$

Condition 2 means that if σ and τ are incompatible, then $T(\sigma)$ and $T(\tau)$ are incompatible if defined. If T is a total function, then the tree is described as *perfect*.

Although not strictly correct, we follow standard conventions in terms of the following abuses of notation. We say a string τ is in, or lies in, T if τ is in the range of T. We denote this $\tau \in T$. Given an infinite binary sequence $A \subset \omega$ (possibly the characteristic function of a set) we say that it is a branch of T if for infinitely many $\tau \subset A$, $\tau \in T$. If this is the case we may also say that A lies on T or (for emphasis) we might refer to A as an infinite branch of T.

T_0 is a subtree of T_1 if it is a tree and the range of T_0 is a subset of the range of T_1. We denote this $T_0 \subseteq T_1$. τ is of level n (or of height n) in a tree T if $\tau = T(\sigma)$ for some σ of length n. τ is a leaf of T if $\tau \in T$ and there do not exist any proper extensions of τ in T. A tree is of height n if it is finite and its leaves are at most of height n.

A tree is computable if it is computable as a function $2^{<\omega} \to 2^{<\omega}$. This means that the range of a computable tree is a computably enumerable subset of $2^{<\omega}$. Given a computable tree T, we let $T[s]$ denote the portion of the tree enumerated by stage s. We assume that if $\sigma \subset \tau$ and $T(\tau)\downarrow$, then $T(\sigma)$ is enumerated before $T(\tau)$. When talking about trees we say that τ is a successor of σ if both σ and τ lie in T and $\sigma \subset \tau$. We may also say that σ is a predecessor of τ. If there is no string γ with $\sigma \subset \gamma \subset \tau$ we say that τ is an immediate successor of σ, or that σ is the immediate predecessor of τ. In general we use the notation σ^- to mean the immediate predecessor of σ.

1.3 Priority Arguments

Priority arguments are ubiquitous in computability theory - many proofs involve them in some form. The basic idea behind the priority method is to break down the property (or properties) the set being constructed must have into infinitely many requirements, and give them a priority ordering. As an example, if we require that $A \neq \Psi_i$ for all i then A will be incomputable. In a standard priority argument we give a strategy to satisfy each requirement in isolation, and then give a method for combining these strategies so that all of the requirements are satisfied. We usually start by giving the ideas naively, before formalising the construction. The action of given strategies may interact and if two strategies have competing (or conflicting) aims to satisfy their requirements, then precedence is given to the requirement of higher priority. Once the priority argument is completed, and the required sets constructed, we have a verification (usually using an inductive argument) which shows that the set(s) constructed do indeed have the required properties.

Friedberg and Muchnik independently came up with a construction which allowed requirements of higher priority to injure requirements of lower priority. They did this to prove

Theorem 1.3.1 *[Fri57];[Muc56] There exist incomparable c.e. degrees.*

We first describe in general how a tree of strategies works, and then provide a proof of the Friedberg-Muchnik Theorem using a tree argument, (which we split into a naive construction, formal construction and verification) by way of example.

The simplest tree of strategies is a downwards closed subset of $n^{<\omega}$, for some $n \in \omega$ - though they may also be downwards closed subsets of $\omega^{<\omega}$ or $(\omega + n)^{<\omega}$. To create a tree of strategies, as well as the downwards closed set, we need a way of assigning a requirement to each node. Formally a node is a node on the tree of strategies and a module is the version of a requirement associated with a node. In practice, when discussing the tree, we use these terms interchangeably. We use calligraphic letters to denote a generic requirement on the tree of strategies, and lower case Greek letters to give examples of specific modules. The branches leaving a node are its external states. Nodes may also have a set of internal states, these combined with the external states give the modules possible outcomes.

Each module has a set of instructions to perform and which instructions it performs will depend upon the state that it is in. At the beginning of a stage control is passed to the module at the base of the tree. This module then acts based upon its current state and its instructions. Control is passed to the immediate successor of the module, along the branch corresponding to the modules outcome. In this situation, for brevity, we normally omit the word 'immediate'. The module that is passed control then acts, according to its instructions and so the process continues. We bound the height of the tree of strategies which can be reached at any given stage (so as to ensure that we only attempt to access finitely many modules at each stage) by ending a stage s when height s is reached on the tree of strategies. A stage may finish before this happens, if instructed to by a module accessed during the stage. If a module performs no actions, then it simply passes control along the branch according to the outcome it was in when passed control. We define the *control path* for a given stage (CP_s) to be the set of modules passed control during the stage. That is, if σ is the last module to act in a given stage, then the control path for that stage is the set of modules on substrings of σ. Naively the *true path* (TP) is the string on the tree of strategies which builds the required sets. How we formally define the true path varies from construction to construction. In the Friedberg-Muchnik example it is the rightmost path visited during the construction, though this is not always the case.

Proof (of Theorem 1.3.1):

We build two c.e. sets A and B such that $A \not\leq_T B$ and $B \not\leq_T A$. To do this we build approximations A_s and A_s, such that $A_s \subseteq A_{s+1}$ and $B_s \subseteq B_{s+1}$, to A and B respectively. We end by defining $A = \bigcup_s A_s$ and $B = \bigcup_s B_s$. We start with $A_0 = B_0 = \emptyset$ and at stage s use the current approximation to A or B when performing calculations. We have the following requirements.

$\mathcal{Q}_i : \ A \neq \Psi_i(B)$

$\mathcal{R}_i : \ B \neq \Psi_i(A)$.

For an individual requirement the strategy is simple. $\mathcal{Q}_i$ picks a witness x (not yet in A) to try and show that $A(x) \neq \Psi_i(B,x)$. $\mathcal{R}_i$ does nothing until we reach a stage $s+1$ where $\Psi_i(B_s,x)\downarrow[s] = 0$, at which point it enumerates x into A_s. There are two possible outcomes here, either $\Psi_i(B,x)\uparrow$ (but $A(x)$ is defined), or $\Psi_i(B,x)\downarrow\neq A(x)$ - both of which satisfy the requirement. (The strategy for a $\mathcal{R}_i$ requirement is the same, but with A and B switched). The difficulty here is that a requirement may see $\Psi_i(A,x)\downarrow= 0$ at some stage and act accordingly (enumerating x into B), but then at some later stage a requirement of higher priority alters B below $u(A,i,x)$. This means that we can no longer be certain of the final value of $\Psi_i(A,x)$. We use the tree of strategies to solve this problem.

Formally a $\mathcal{Q}_i$ module has two possible states - labeled 0 and 1. In state 1 it has managed to successfully diagonalise A and $\Psi_i(B)$. In this case this particular module will not act

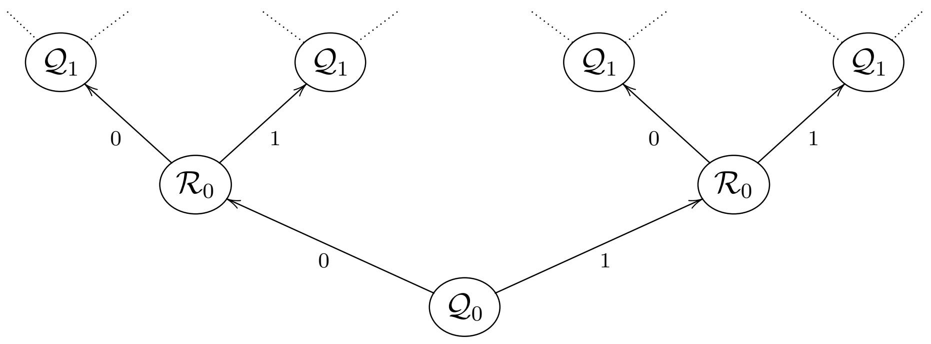

Figure 1.1: Friedberg-Muchnik Tree of Strategies

again, though a change of state lower down the tree may cause another module associated with the same requirement to be accessed, and the diagonalisation reattempted. At stage s in state 0, if the module does not have an associated value, then it picks n such that $A_{s-1}(n) \uparrow$ and sets $A_s(n) \downarrow= 0$. If the module does have an associated n then it checks if $\Psi_i(B_{s-1}, n)[s-1] \downarrow$. If so and $\Psi_i(B_{s-1}, n) \neq 0$ (or $A_{s-1}(n)$), then it sets $A_s = A_{s-1}$, $B_s = B_{s-1}$ and enters state one. If $\Psi_i(B_{s-1}, n) = 0$, then it sets $A_s = A_{s-1} \cup \{n\}$, $B_s = B_{s-1}$ and enters state one.

An $\mathcal{R}_i$ module acts precisely the same, but with the roles of A and B swapped.

In Figure 1.1 you can see the lay out of the base few nodes in the tree of strategies. If we consider the left most Q_1 module. When this module is first passed control it picks a witness, and then at some later stage, s, it acts to diagonalise using some B value. If at some stage $s' > s$, after this has happened, the left most $\mathcal{R}_0$ module acts, altering the B value that $\mathcal{Q}_1$ used, however in doing so $\mathcal{R}_0$ moves into state 1, and a different $\mathcal{Q}_1$ module now attempts to diagonalise $\Psi_1(B)$ and A. This new module is able to pick a new witness, and wait for a new β value which allows it to diagonalise. If no such β is everfound then $\Psi_1(B)$ is partial, and so not equal to A.

The discussion above can be condensed into the following formal construction

Stage 0*:* Set $A_0 = \lambda$ and $B_0 = \lambda$.

Stage $s + 1$*:* We are given A_s and A_s. We describe the action of $\mathcal{Q}_i$ nodes, for $\mathcal{R}_i$ nodes the action is the same but with A_s and B_s swapped. Start at the bottom of the tree and work upwards as follows. The stage finishes when height s of the tree is reached, unless instructed to before that height is reached. If a node is in state 1, do nothing and move to the next node on the control path. If a node is in state 0, and does not have an associated value, pick $n \geq s$ larger than previously used and larger than the use of any previously computed functional, associate it with the node and set $A_{s+1}(n) = 0$. If the node is in state 0, and has associated value n, check if $\Psi_i(B_s, n)[s] \downarrow \neq A_s(n)$. If so set $A_{s+1} = A_s$, move the node into state 1 and finish the stage. If not, and $\Psi_i(B_s, n) \downarrow$, set $A_{s+1} = A_s \cup \{n\}$, move the module into state 1 and finish the stage. If $\Psi_i(B_s, n) \uparrow$ move to the next node on the control path.

Formally we verify that all the requirements are satisfied as follows. We argue without loss of generality for the $\mathcal{Q}$ module case - the arguments all work in the $\mathcal{R}$ case, with $\mathcal{Q}$ and $\mathcal{R}$ swapped and A and B swapped.

First we show that no module higher up the tree of strategies is able to injure one lower down the tree. Let ρ be an $\mathcal{Q}_i$ module with witness n_ρ. Every $\mathcal{Q}$ module of lower priority (on the same branch as ρ) must have a witness strictly larger than n_ρ, and so no $\mathcal{Q}$ module of lower priority is able to injure ρ by enumerating n_ρ in A. We must also consider the actions of lower priority $\mathcal{R}$ modules. Suppose at stage t, ρ sees $\Psi_i(\alpha_t, n_\rho) \downarrow [t] = 0$ and then at some stage $t' > t$ a $\mathcal{R}_{i'}$ module (with $i' > i$) wants to enumerate into (the set represnted by) α_t below height n_ρ. This is impossible, as when ρ first saw convergence it entered state 1, and so a new $\mathcal{R}_{i'}$ module was accessed and it picked a new witness, larger than previously seen and larger than the use in computing any previously computed functional and hence larger than $u(A_t, i, n_\rho)$.

Now we have to show by induction that all requirements are satisfied. We do this by induction as follows. At stage $s = 0$ the True Path is the empty string and so the base case holds. Let $s' > 0$ be the stage such that all modules of height $< t$ on the True Path (on the tree of strategies) have acted. Let ρ be the module at height t on the True Path, and without loss of generallity assume that ρ is a Q module, with associated functional Ψ_i and associated value n_ρ. No module below or to the left of ρ on the tree of strategies will act after stage s'. Consider two cases:

1. For no $s'' > s'$ do we see $\Psi_i(A_{s''}, n_\rho)[s] \downarrow$. In this case ρ is satisfied as $B(n_\rho) = 0$ and $\Psi_i(A, n_\rho) \uparrow$.
2. There is some stage $s'' > s'$ such that $\Psi_i(A_{s''}, n_\rho)[t'] \downarrow$. If this is the case then at stage $s'' > s'$ diagonalises, so $B_{s''}(n_\rho) \neq \Psi_i(A_{s''}, n_\rho)$. From before we know that after this stage ρ is never again injured and so $B(n_\rho) \neq \Psi_i(A, n_\rho)$

This completes the induction, and hence the proof.

□

In the example above each module can only injure modules of lower priority finitely many times, hence constructions of this type are called *finite injury constructions*. It is possible for a construction to be *infinite injury* as well. When a construction uses an oracle to perform a step (typically, though not necessarily, $\emptyset^n$ for some n) it is called an *oracle construction*. A construction may be described as *full approximation* if it is an (effective) priority construction where the sets being constructed are not recursively enumerated but computably approximated.

Further information on priority arguments and details of using a tree of strategies can be found in [Odi99].

Chapter 2

The Local Degrees

In this chapter we look at the degrees below $\emptyset'$. The main results covered are to do with the High-Low Hieararchy - first described in [Coo74] and [Soa74]. Proofs concerning the High-Low Hierachy come originally fro [Sac63], but we use the method from [JS84]. We also introduce the concept of natural definability. An introduction to this can be found [Lew]. The content of this chapter is not essential for following the new work in Chapters 4 and 5, but helps to give a broader overview of the subject.

The study of the structure of the degrees in $\mathcal{D}$, is a main area of focus in computability and within this those degrees below $\mathbf{0}'$ - known as the local degrees - are of especial interest. We have already seen that a natural ordering exists on $\mathcal{D}$, and that incomparable degrees exist so the ordering cannot be linear.

To see that any pair of degrees has a unique least upper bound note that given any two representative sets A and B from degrees $\mathbf{a}$ and $\mathbf{b}$, we are able to construct a set $A \oplus B$ whose characteristic function is $(A \oplus B)(2n) = A(n)$ and $(A \oplus B)(2n+1) = B(n)$. Any set which computes both A and B computes $A \oplus B$, and $A \oplus B$ computes both and A and B. From this we see that the least upper bound of any two degrees $\mathbf{a}$ and $\mathbf{b}$ is $deg(A \oplus B)$. In Chapter 3, (Theorem 3.2.1) we construct a pair of degrees which has no non-trivial lower bound.

We use the notation $\mathbf{a} \vee \mathbf{b}$ to mean the least upper bound of $\mathbf{a}$ and $\mathbf{b}$; and $\mathbf{a} \wedge \mathbf{b}$ to mean the greatest lower bound of $\mathbf{a}$ and $\mathbf{b}$, if it exists.

When looking at structural properties and definability within $\mathcal{D}$ another important area of study is *natural definability* (see for instance [Lew]). What exactly we mean by natural definability is not clear, as the notion of what makes a mathematical statement *natural* is clearly not a precise one. In Section 2.2 we will consider more what we mean by natural.

2.1 The High-Low Hierarchy

In this section we discuss the *High-Low Hierarchy*, as developed by Cooper [Coo74] and Soare [Soa74]. It gives us a way of describing the information content of a degree below $\mathbf{0}'$. Informally the higher a degree is, the closer to $\mathbf{0}'$ it is and the more 'knowledge' (in some sense) it has; conversely the lower a degree is the closer it is to $\mathbf{0}$ and the less 'knowledge' it has.

Definition 2.1.1 *For $n \geq 1$ we have the following definition of the High-Low Hiearchy.*

$$high_n = \{\mathbf{a} \leq \mathbf{0}' : \mathbf{a}^n = \mathbf{0}^{n+1}\}$$

$$low_n = \{\mathbf{a} \leq \mathbf{0}' : \mathbf{a}^n = \mathbf{0}^n\}$$

.

Given a set A if $deg(A) \in high_n$ we say that A and $\mathbf{a} = deg(A)$ are $high_n$, sometimes shortened to H_n. Similarly for low_n. If $n = 1$ we often drop the subscript and simply say that a set or degree is *low* or *high*.

That the high-low hierarchy does not collapse follows from results in [Sac63]. We give a proof (in Theorem 2.1.2), which follows the ideas given in [JS84]. First we use a full approximation construction to build a low degree. We use a full approximation construction, rather than the easier oracle construction, as we are later able to give simple modifications which allow us to construct a set which is low relative to an arbitrary degree. Once we have constructed a low degree we introduce the psuedo-jump operator and then prove a theorem which when combined with the low degree construction will allow us to build sets which are $high_{n+1}$, but not $high_n$, and low_{n+1}, but not low_n for all n.

Theorem 2.1.2 *(From work in [Sac63])There exists a set $A \neq_T \emptyset$ which is c.e. and low.*

Proof:[JS84] We construct a set A using a basic priority argument, such that A is c.e. and $deg(A)' = 0'$. We build A in stages, and ensure that $A_s \subseteq A_{s+1}$ - where A_s is the approximation to A at stage s. In the end we will define $A = \bigcup_s A_s$.

Outline: We have the following two requirements:

$$\mathcal{P}_i : \ W_i \text{ infinite} \Rightarrow W_i \cap A \neq \emptyset,$$

$$\mathcal{Q}_i : \ \exists^\infty s[\Psi_i(A_s, i)[s] \downarrow] \Rightarrow \Psi_i(A, i) \downarrow.$$

Where $\exists^\infty$ means "there exist infinitely many". We must also ensure that A is coinfinite, and so (with the $\mathcal{P}_i$ requirements) that A is simple.

The Q_i requirements look to show that A is low. This is done in two parts, first consider the function

$$g(i,s) = \begin{cases} 1 & \text{if } \Psi_i(A_s, i)[s] \downarrow \\ 0 & \text{otherwise} \end{cases}$$

and let $g^*(i) = lim_s g(i,s)$, which exists if Q_i is satisfied. g^* is the characteristic function of A', and as g is computable, g^* can be computed using an oracle for $\emptyset'$ - ensuring that A is low.

In order to satisfy the Q_i requirements we make use of a restraint function, which allows us to stop the construction injuring Q_i requirements which are of 'too high' priority - exactly how this works is described later. We define the restraint function as follows:

$$r_i(s) = u(A_s, i, i, s).$$

The restraint function is injured at stage $s+1$ of the construction if some $n < r_i(s)$ is enumerated into A. Then if there is some stage after which r_i is not again injured then Q_i is satisfied and $lim_s r_e(s)$ is defined. To see this let s_0 be the least stage after which r_e is never again injured. As for all stages $t \geq s_0$ if $\Psi_i(A_t, i)[t] \downarrow$ then Q_i is satisfied and in the limit $r_i(s)$ equals 0. Alternatively let $t \geq s_0$ be the least subsequent stage at which $\Psi_i(A_t, i)[t] \downarrow$. No numbers are enumerated into A which are less than $u(A_t, i, i, t)$ after stage t, and so the computation is preserved with $\Psi_i(A, i) \downarrow$. Hence $r_i(s) = r_i(t)$ for all $s \geq t$.

The P_i requirements, combined with making sure that A is coinfinite, ensure that the complement of A is not equal to W_i for any i - this is sufficient to guarantee that A is incomputable, as A is computable if and only if both A and $\bar{A}$ are c.e.

Ordering the Requirements: The requirements are given the following priority order

$$Q_0 > P_0 > Q_1 > P_1 > \cdots > Q_n > P_n > \cdots$$

Where $A > B$ denotes "A is of higher priority than B". Q_0 is the module of highest priority. No module is allowed to injure a module of higher priority. Specifically P_i may only enumerate n into A at stage $s+1$ if for all $j \leq i$, $n > r_j(s)$. Each P_i module need act only once, as once it has enumerated a number into A it is irreversibly satisfied.

All modules will eventually be satisfied as each Q_i module is of lower priority than only finitely many P_i modules. Once all the P_i modules below it are satisfied (which occurs at some finite stage) the Q_i module is never again injured. As each Q_i module is satisfied, $lim_s r_i(s)$ exists for all i. All P_i modules are satisfied as well, as if W_i is infinite then it will have a member which is greater than the restraint function of all modules of higher priority - this can be enumerated into A as required. If W_i is finite then the implication is automatically satisfied.

Construction: The formal construction is as follows

Stage 0*:* Set $A_0 = \emptyset$

Stage $s+1$*:* Given A_s, find the least $i \leq s$ (if such an i exists) such that:

1. $W_{i,s} \cap A_s = \emptyset$,
2. $\exists x[x \in W_{i,s} \wedge x > 2i \wedge (\forall e \leq i)[r_e(s) < x]]$.

If such an i exists then pick the least such i and enumerate the least x satisfying (2) into A and go to the next stage. If not, go to the next stage.

Verification: $\bar{A}$ is infinite as each P_i requirement may enumerate at most one element into A. If it does enumerate an element x into A then $x > 2i$.

The formal induction that every requirement is satisfied uses the arguments given in the "Ordering the Requirements" section and is omitted.

This completes the proof of Theorem 2.1.2.

$\square$

We can relativise the proof of lowness as follows:

Definition 2.1.3 *A set A is low relative to a set Y if $A \geq_T Y$ and $A' =_T Y'$.*

This extends to A being low_n relative to Y and transfers to A being $high_n$ relative to Y in the obvious way.

Showing the existence of a set which is low relative to a set Y is very simliar to showing the existence of a set which is low. We construct a set A which is c.e. in Y. The construction follows the same basic pattern, except that we ensure that A is not computable in Y and when constructing g and g^* we ensure that g is computable in Y and that g^* can be computed using an oracle for Y'. This guarantees that $Y \oplus A$ has the desired properties. Full details are omitted.

Next we introduce the *pseudo-jump operator* (first described by Jockusch and Shore [JS83]) which naively acts like a "little jump operator".

Definition 2.1.4 *We define $J_e(Y)$ to be the e^{th} pseudo-jump of Y. For $e \in \omega$ and $Y \subseteq \omega$ define $J_e(Y) = Y \oplus W_e^Y$*

We define the pseudo-jump operator to be $Y \oplus W_i^Y$ so as to ensure that $Y \leq_T J_i(Y)$. We now show that there exists a set which pseudo-jumps to $\mathbf{0}'$.

Theorem 2.1.5 *[JS83]For every i there is a non-computable c.e. set A such that $J_i(A) \equiv_T \emptyset'$*

Proof: Fix i. We construct a c.e. set A, such that $deg(A) > \mathbf{0}$ and such that, $deg(A \oplus W_i^A) \geq \mathbf{0}'$. We build A in stages, and ensure that $A_s \subseteq A_{s+1}$. In the end we will define $A = \bigcup_s A_s$.

Outline: To prove this we have three types of requirements, the first aiming to show that $W_i^A \leq_T \emptyset'$, the second guarunteeing that A is not computable and the aiming to show that $A \oplus W_i^A \geq \emptyset'$.

$\mathcal{P}_{2x} : \; x \in \emptyset' \Leftrightarrow (\exists y \leq h(2x))[y \in \omega^{[2x]}$ and $y \in A]$,

$\mathcal{P}_{2x+1} : \; A \neq \bar{W}_e$

$\mathcal{Q}_x : \; (\exists^\infty s)[x \in W_{i,s}^{A_s}] \Rightarrow x \in W_i^A.$

$h(x)$ is defined formally later but the basic idea for the $\mathcal{P}_{2x}$ requirements is as follows. We wish to encode $\emptyset'$ into $A \oplus W_i^A$. To do this we first ensure that $A \cap \omega^{[x]}$ is non-empty if $x \in \emptyset'$ (i.e., we enumerate an element into the x^{th} column of A iff $x \in \emptyset'$), and then we ensure that there exists some function $h \leq_T W_i^A$ such that whenever there is a number in the x^{th} column of A, it is less than $h(2x)$. In this way we ensure that $\emptyset' \leq_T A \oplus W_i^A$ - using arguments similar to those used previously we are able to build characterstic function of $\emptyset'$ which is computable in $A \oplus W_i^A$.

The $\mathcal{Q}_x$ requirements follow the same pattern as the $\mathcal{Q}$ requirements in the previous proof. As before we build a function computable given an oracle for $\emptyset'$ which allows us to compute the characteristic function of $A \oplus W_i^A$, and use a restraint function to ensure that all the $\mathcal{Q}$ requirements are satisfied. Consider the following function,

$$g(i,s) = \begin{cases} 1 & \text{if } x \in W_{i,s}^{A_s} \\ 0 & \text{Otherwise} \end{cases}$$

Arguments given in the previous proof ensure that if the $\mathcal{Q}$ requirements are satisfied then $g^*(i) = lim_s g(i,s)$ exists, is computable in $\emptyset'$ and computes $A \oplus W_i^A$. We define the restraint function as follows:

$$r_x(s) = u(A_s, i, x, s).$$

The restraint is injured if at some stage an element is enumerated into A which is less than the current value of the restraint function for x. As before we require (for each x) that after some finite stage of the construction the restraint is never again injured.

Ordering the Requirements The requirements are given the following priority order

$$\mathcal{Q}_0 > \mathcal{P}_0 > \mathcal{Q}_1 > \mathcal{P}_1 > \cdots > \mathcal{Q}_n > \mathcal{P}_n > \cdots$$

With $\mathcal{Q}_0$ being the module of highest priority. No module is allowed to injure a module of higher priority.

Each $\mathcal{P}_x$ module only has finitely many $\mathcal{Q}_{x'}$ modules which are of higher priority than it. If we have an oracle for W_i^A then we know which x' are in W_i^A, and using an oracle for A we are then able to compute $lim_s r_{x'}(s)$. So using an oracle for $A \oplus W_i^A$ we are able to find a position within the x^{th} column after which enumerating numbers will not injure requirements of higher priority.

Construction:

Stage 0*:* Set $A_0 = \emptyset$. For all x define $h^*(x, 0) = \langle x, 0\rangle$

Stage $s+1$*:* At stage $s+1$, given A_s, define the computable function

$$h^*(x,s) = (\mu y)\Big[y \in \omega^{[x]} \wedge h^*(x-1,s) < y \wedge h(x,s-1) \leq y \wedge (\forall j \leq x)[r_j(s) < y]\Big].$$

Given A_s we have $r_j(s)$ for all j.

Fix an enumeration $\{\emptyset'_s\}$ of $\emptyset'$, and consider the P_{2x} and $\mathcal{P}_{2x+1}$ requirements serparately

Choose the least $j \leq s$ such that $\emptyset'_{j+1} - \emptyset'_j \not\subseteq \emptyset$ and $[\emptyset'_{j+1} - \emptyset'_j] \notin A_s$ i.e., such that the element enumerated into $\emptyset'$ at the j^{th} step of our fixed enumeration is not in our approximation to A at stage s. If such a j exists choose the least $x \in \emptyset'_{j+1} - \emptyset'_j$ and enumerate $h^*(2x, s)$ into A_{s+1}. If such a j exists we say that P_{2j} receives attention. If such an j exists we say that $\mathcal{P}_{2j}$ receives attention. $\mathcal{P}_{2j}$ is satisfied for all stages >s+1 and hence $\mathcal{P}_{2j}$ never again receives attention. If i does not exist, do nothing.

Secondly choose the smallest j such that $j \leq x$, $W_{j,s} \cap A_s = \emptyset$ and

$$(\exists x)\Big[h(2x+1, s) \in W_{j,s} \;\&\; x > 2j \;\&\; (\forall e \leq j)[r(e, s) < x]\Big]$$

If such a j exists choose the least x satisfying the above equation. Enumerate this into A_{s+1}. If this happens we say that $\mathcal{P}_{2j+1}$ receives attention. $\mathcal{P}_{2j+1}$ is satisfied for all stages >s+1 and hence $\mathcal{P}_{2j+1}$ never again receives attention. If j does not exist, do nothing.

We define $h(x) = lim_s h^*(x, s)$.

Verification

We say that y injures $\mathcal{Q}_x$ at stage $s + 1$ if $y \in A_{s+1} - A_s$ and $y \leq r_j(s)$. We define the injury set set for $\mathcal{Q}_x$ as follows

$$I_x = \{\text{y: y injures } \mathcal{Q}_x \text{ at some stage } s+1\}$$

We now have to show that our construction does indeed satisfy all of the requirements, required in the proof of Theorem 2.1.5.

Lemma 2.1.6 *For all x, I_x is finite.*

Proof: (Note: This is the same as saying that for each x, $\mathcal{Q}_x$ is injured at most finitely often.) Each positive requirement contributes at most 1 element to A. Also, $\mathcal{Q}_x$ can be injured by $\mathcal{P}_{x'}$ only if $x' < x$. So $|I_x| \leq x$.

□

Lemma 2.1.7 *For every x, $\mathcal{Q}_x$ is met and $r(x) = lim_s r_x(s)$ exists.*

Proof: Fix some x. By the previous Lemma we know that there is an s such that $\mathcal{Q}_x$ is not injured at any stage $s' > s$. If $x \in W_{x,s}^{A_s}$ for $s' > s$ then (by induction on $t \geq s$) $r_x(t) = r_x(s)$ and $W_{x,t}^{A_t}(x) = W_{x,s}^{A_s}(x)$ for all $t \geq s$ so $A_s \upharpoonright_r = A \upharpoonright_r$ for $r = r_x(s)$ and hence $W_x^A(x)$ is defined.

□

Lemma 2.1.8 *For all j, $h(j) = lim_s h^*(j, s)$ exists.*

Proof: This follows from the definition of $h^*(j, s)$ and Lemma 2.1.7.

□

Lemma 2.1.9 *For every x, requirement $\mathcal{P}_x$ is met.*

Proof: Fix some j.

If $j = 2x$: Choose s such that $(\forall t \geq s)(\forall e \leq x)[r_e(t) = r(t)]$. Choose $s' \geq s$ such that no $\mathcal{P}_{j'}$, $j' < j$ receives attention after stage s'. Choose $t > s'$ such that

$$\exists x \Big[x \in \emptyset'_{x+1} - \emptyset'_x \text{ and } (\forall e \leq x)[r(t) < x] \Big]$$

Now, either $h(j, t) \in A_s$ or $\mathcal{P}_j$ receives attention at the next stage i.e., j is enumerated into a column of A at the next stage. Either way $\mathcal{P}_j$ is met.

If $j = 2x + 1$. Choose s such that $(\forall t \geq s)(\forall e \leq x)[r(e, t) = r(e)]$. Choose $s' \geq s$ such that no P_j, $j' < j$ receives attention after stage s'. Choose $t > s'$ such that

$$\exists x\Big[x \in W_{j',t} \text{ and } (\forall e \leq j'[r(e) < x]\Big]$$

Now, either $W_{j,t} \cap A_t \neq \emptyset$ or P_j receives attention at the next stage. Either way P_j is met.

$\square$

Now note that $r \leq A \oplus W_i^A$, and hence $h \leq A \oplus W_i^A$. As noted earlier this gives us that $\emptyset' \leq_T A \oplus W_i^A$, and so completes the proof.

$\square$

This concludes the proof of Theorem 2.1.5

The proof of Theorem 2.1.5 is uniform in e and can be relativised to any oracle. That is there exist computable functions f and g such that for all $e \in \omega$ and $Y \subseteq \omega$

$$J_e(J_{f(e)}(Y)) = Y', \text{ via } g(e).$$

i.e.,

$$J_{f(e)}(Y) \not\leq_T Y \text{ and } Y = \Psi_{g(e)}^{J_{f(e)}(Y)}.$$

Also, note that: Y' is low_n over $C \Leftrightarrow Y$ is $high_n$ over C.

Finally we come to the main proof, that the high-low hierachy doesn't collapse.

Theorem 2.1.10 *([Sac63])For every n there exist c.e. degrees in $\boldsymbol{H}_{n+1} - \boldsymbol{H}_n$ and $\boldsymbol{L}_{n+1} - \boldsymbol{L}_n$*

Proof: We prove this by induction.

Note that a set Y' is low_n over C if and only if C is $high_n$ over Y.

The existence of a low_1 set is given in Theorem 2.1.2. Theorem 2.1.2 can be relativised to give an index e_1 for which $Y <_T W^Y_{e_1}$ and $Y' \equiv_T (W^Y_{e_1})'$, i.e., W_{e_1} is low relative to Y. We can use Theorem 2.1.5 on index e_1 to produce a set A which $\emptyset'$ is relative to, and hence which is $high_1$. This completes the base of the induction.

Assume that the theorem holds for n. We wish to show that it hold for $n+1$. At stage n we were an index, e_n say, such that for any set Y, $W^Y_{e_n}$ is $high_n$ over Y. We can use the relativisation of Theorem 2.1.2 on e_n to produce a set A which is low_{n+1} and then can use Theorem 2.1.5 to construct a set B such that $\emptyset'$ is low_{n+1} relative to B' and hence B is $high_{n+1}$.

This finishes the induction, and hence the proof.

□

The high-low hierarchy gives us a horizontal layering of the local degrees, and is important as many properties of the local degrees are dependent on the jump class a degree lies in (i.e., where in the high-low hierarchy a degree lies). For instance the minimal degrees we construct in Chapter 3 are all low_2 and the 1 generic degrees we are concerned with in Chapter 5 are low_1.

It is possible to generalise the concept of the jump classes to the Turing degrees as a whole (see [JP78]), using the following definition.

Definition 2.1.11 *For* $n \geq 1$ *a degree* $\mathbf{a}$ *is* generalised low$_n$ *if* $\mathbf{a}^n = (\mathbf{a} \vee \mathbf{0}')^{n-1}$. *A degree* $\mathbf{a}$ *is* generalised high$_n$ *if* $\mathbf{a}^n = (\mathbf{a} \vee \mathbf{0}')^n$.

Intuitively a degree is low_n if its n^{th} jump is as low as it can be with relation to $\mathbf{0}'$ and *generalised low*$_n$ if its n^{th} jump is as low as it can be with relation to $\mathbf{a} \vee \mathbf{0}'$; and a degree is $high_n$ if its n^{th} jump is as high as it can be with relation to $\mathbf{0}'$ and *generalised* $high_n$ if its n^{th} jump is as high as it can be with relation to $\mathbf{a} \vee \mathbf{0}'$. Below $\mathbf{0}'$ the low and generalised low classes, and high and generalised high classes coincide.

Many results from the high-low hierarchy generalise but this is not always the case. An important example is that every degree which is not above $\mathbf{0}'$ is bounded by a generalised low degree. As there exist generalised high degrees below $\mathbf{0}'$, the generalised jump classes do not respect our ordering on the Turing degrees in the same way the (non-generalised) jump classes do.

2.2 Natural Definability

Within the Turing Degrees a relationship is *definable* if there is a formula in the language of partial orders which is true of those tuples of degrees in the relationship.

Definability results do exist for some of the key concepts in computability. $\mathbf{0}'$ was shown to be definable in [SS99], the definition of the jump function is definable by a relativisation of this, and Nies, Shore and Slaman [NSS98] showed that all jump classes except low are definable in the c.e. degrees.

The technique used in these proofs (and in many other definability results) is to encode models of arithmetic into the degree structure. Although a powerful technique it does not yield what would be considered *natural* definitions. The question left is whether the presented definition of a given relation (or one which is equally unnatural) is the only one which exists or whether there might be a more natural one.

What exactly is natural is not precise, but intuitively we wish the definition to be in the language of partial orders, relatively short, and contain relatively few alterations of quantifiers. Such a definition might be of the form $\mathbf{a}$ is the least degree such that all degrees above have a certain property or the greatest degree such that all degrees below have a certain property, or some other equally 'simple' concept.

2.3 Properties in the Local Degrees

In this section we give definitions for some properties which a degree may have, and are potentially useful in our search for natural definability results. For this to happen we need the properties to themselves have some natural definition. We think of an order theoretic property being natural if the property can be described by a formula in the first order language for Turing degrees and which uses only a small number of non-logical symbols. As before we require the definition to be not too long, or have too many alterations of quantifiers.

In Chapter 4 and Chapter 5, we will give results about certain classes of degrees (within the local degrees) concerning the following properties.

Definition 2.3.1 *A degree* $\mathbf{a} > \mathbf{0}$ *satisfies the* join property *if, for all non-zero* $\mathbf{b} < \mathbf{a}$ *there exists* $\mathbf{c} < \mathbf{a}$ *with* $\mathbf{b} \vee \mathbf{c} = \mathbf{a}$.

Definition 2.3.2 *A degree* $\mathbf{a} > \mathbf{0}$ *satisfies the* meet property *if, for all* $\mathbf{b} < \mathbf{a}$ *there exists non-zero* $\mathbf{c} < \mathbf{a}$ *with* $\mathbf{b} \wedge \mathbf{c} = \emptyset$.

We can extend the previous two definitions as follows

Definition 2.3.3 *A degree* $\mathbf{a}$ *satisfies the* complementation property *if for all non zero* $\mathbf{c} < \mathbf{a}$ *there exists non-zero* $\mathbf{b} < \mathbf{a}$ *such that* $\mathbf{b} \vee \mathbf{c} = \mathbf{a}$ *and* $\mathbf{b} \wedge \mathbf{c} = \mathbf{0}$.

These definitions satisfy the conditions we gave to be natural and $\mathbf{0}'$ satisfies all of them [Pos81]. It is however not known whether $\mathbf{0}'$ is the least degree such that all degrees above satisfy complementation. A positive answer to this would be a natural definition for $\mathbf{0}'$.

In this thesis we are looking at order theoretic properties concerning meet and join, in the hope that they will provide some use in helping to show whether $\mathbf{0}'$ is indeed the least such degree.

We show in Chapter 3 (Theorem 3.2.1) that degrees exist which satisfy the following definition.

Definition 2.3.4 *A degree* $\mathbf{a} > \mathbf{0}$ *is minimal if* $\mathbf{b} < \mathbf{a}$ *implies that* $\mathbf{b} = \mathbf{0}$.

This shows us that the local degrees are not densely ordered. In Chapter 4 we will use this property to show certain classes of degrees satisfy the meet property.

The final result in this thesis (Theorem 5.2.2) concerns *generic degrees*. Our notion of generic degrees comes originally from Cohen's idea of forcing, invented by him in 1963 [Coh63]. In 1965 Feferman showed that one could force relative to the first order language of arithmetic [Fef65] and in 1977 Jockusch developed the version of forcing we currently use in computability theory [Joc77b]

Definition 2.3.5 *A set A is* 1-generic *if for every c.e. set of strings W either*

1. $(\exists\sigma \subset A)[\sigma \in W]$*; or*
2. $(\exists\sigma \subset A)(\forall\tau \supseteq \sigma)[\tau \notin W]$.

A degree is 1-generic if it contains a 1-generic set. In Chapter 5 (Theorem 5.0.5) we show how to build 1-generic sets within the local degrees.

Chapter 3

Minimal Degree Constructions

In this chapter we cover minimal degree constructions. A Turing degree, $\mathbf{a}$, is minimal if $\mathbf{a} \neq \mathbf{0}$ and for any degree $\mathbf{b}$ if $\mathbf{b} < \mathbf{a}$ then $\mathbf{b} = \mathbf{0}$. The first minimal degree was constructed by Spector in [Spe56]. In [Sac61], Sacks adapted the proof given by Spector, to fit within the ideas given by Friedberg and Muchnik in order to show that there exists a minimal degree $\mathbf{a} \leq \mathbf{0}'$. In [Sho66] Shoenfield, showed that if $\mathbf{0} < \mathbf{a} < \mathbf{0}'$, then there is a minimal degree $\mathbf{b}$ below $\mathbf{0}$ and incomparable with $\mathbf{a}$. The method he used to construct the minimal degree $\mathbf{b}$ was (presentationally)a considerable simplification of the previous methods used to construct a minimal degree. This framework became the standard one for constructing minimal degrees, and our constructions follow this. We end the chapter with a full approximation construction of a minimal degree, following the ideas of Cooper [Coo72]

The necessary machinery for our construction of minimal degrees is *splitting trees*, and we introduce these first. With this in place we build a minimal degree below $\mathbf{0}''$ and then below $\mathbf{0}'$. Finally we discuss building a minimal degree by full approximation. It is an adaptation of this method which we will use for the results in Chapter 4.

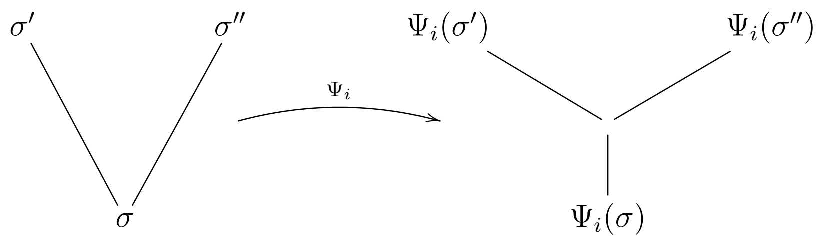

Figure 3.1: A Ψ_i splitting above a string σ

3.1 Splitting Trees

A key concept in the construction of minimal degrees is that of *splitting trees*. Where we are considering binary versions of trees, as defined in Chapter 1.

Given two strings σ and τ, we say they are Ψ_i splitting if neither $\Psi_i(\sigma) \subseteq \Psi_i(\tau)$ nor $\Psi_i(\tau) \subseteq \Psi_i(\sigma)$. We often refer to a pair of Ψ_i splitting strings as "a Ψ_i splitting". So, to "search for a Ψ_i splitting" means to search for two strings which are Ψ_i splitting.

As depicted in Figure 3.1, if asked to find a Ψ_i splitting above a string σ, we search for two strings σ' and σ'', extending σ such that for some n, $\Psi_i(\sigma'; n) \downarrow \neq \Psi_i(\sigma''; n) \downarrow$.

Using the idea of splitting strings we have the following definitions.

Definition 3.1.1 *A tree T is a Ψ_i* splitting tree *if any two incompatible strings on T are also Ψ_i splitting.*

Definition 3.1.2 *A tree T is a Ψ_i* nonsplitting tree *if no pair of strings in T are Ψ_i -splitting.*

A tree may be neither Ψ_i splitting nor Ψ_i nonsplitting. Remembering our convention that Ψ_0 is the identity functional, we can see that $2^{<\omega}$ is a Ψ_0 splitting tree. Given a tree T, a Ψ splitting subtree T' is a subtree of T which is also a Ψ splitting tree. Given a tree T and a branch τ in T, a Ψ splitting subtree above τ is a subtree T' of T which is Ψ splitting and for which $T'(\emptyset) = \tau$.

If we wish to build a set M of minimal degree, we require that it is incomputable and that for all i if $\Psi_i(M)$ is total then it is computable or $M \leq_T \Psi_i(M)$, as this ensures that M satisfies the definition of minimality. To this end we have the following two lemmas.

Lemma 3.1.3 *If M lies on T which is a partial computable Ψ_i splitting tree and $\Psi_i(M)$ is total, then $M \leq_T \Psi_i(M)$.*

Proof: We suppose that M lies on T, a partial computable Ψ_i splitting tree, and that $\Psi_i(M)$ is total. We show that in this case, given an oracle for $\Psi_i(M)$ we can compute M.

To do this we start at the base of T and work upwards. As, by assumption, M lies on T we know that $T(\emptyset) \downarrow\subset M$, and that one of $T(0)$ and $T(1)$ must be an initial segment of M. Compute $T(0) = \tau_0$ and $T(1) = \tau_1$. As M lies on T these values must be defined. T is Ψ_i splitting, so $\Psi_i(\tau_0)$ and $\Psi_i(\tau_1)$ are incompatible, and exactly one of them is compatible with $\Psi_i(M)$. Determine which of them it is and hence which of τ_0 and τ_1 is an initial segment of M.

The process described above used the knowledge of which string on level 1 of the tree was an initial segment of M and gave us the string on level 2 which is an initial segment of M. It is clear how we continue to iterate this process, and so compute M using an oracle for $\Psi_i(M)$.

$\square$

Lemma 3.1.4 *If M lies on T which is partial computable and Ψ_i nonsplitting, then $\Psi_i(M)$ is computable if total.*

Proof: We suppose that M lies on T, a partial computable Ψ_i nonsplitting tree, and that $\Psi_i(M)$ is total. To compute $\Psi_i(M;n)$, search until $\tau \in T$ is found with $\Psi_i(\tau;n)\downarrow$. Such a string exists as M lies on T and $\Psi_i(M)$ is total. Then $\Psi_i(\tau;n) = \Psi_i(M;n)$ as T is Ψ_i nonsplitting.

□

If, for each Ψ_i we can ensure that the requirements for either Lemma 3.1.3 or Lemma 3.1.4 are satisfied, then we can ensure that if $\Psi_i(M)$ is total then it is computable or $M \leq_T \Psi_i(M)$. Using a trick first used by Posner [Pos79] we are also able to remove the need for a separate incomputability requirement.

Lemma 3.1.5 *If, for every i, there exists a computable tree T_i such that M lies on T_i and either*

1. *T_i is Ψ_i splitting, or*
2. *there is some initial segment of M in T_i with no Ψ_i splitting extension in T_i*

then M is not computable.

Proof: Assume that M satisfies the conditions in the lemma. If M is computable then given a string σ we may computably test whether σ is an initial segment of M. Consider the Turing functional defined as follows

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

For any $\mu \subset M$, on any input n, $\Psi(\mu;n)\uparrow$, so M does not lie on a Ψ splitting tree. Every infinite tree contains Ψ splittings, and any tree containing M must be infinite - contradicting the existence of M.

□

The previous three lemmas are integral to the following minimal degree constructions. During our discussions we use them without explicit reference. In the formal verifications we reference them as appropriate.

3.2 A Minimal Degree below $0''$

It is comparatively easy to construct a minimal degree below $\mathbf{0}''$ due to the high amount of power we are able to utilise thanks to the oracle. The proof we give, of the original result by Spector[Spe56], follows that given by Epstein in [Eps75]. We discuss the $\mathbf{0}''$ oracle and the $\mathbf{0}'$ oracle in more detail in Section 3.3.3. We leave the details of the ability of the oracle to answer certain questions until then.

Theorem 3.2.1 *[Spe56]There exists a minimal degree.*

Proof[Eps75]: We construct a set M of minimal degree. Thanks to Posner's trick [Pos79], we have only one type of requirement for this proof, which we must satisfy for all $i \in \omega$.

$\mathcal{R}_i$: If $\Psi_i(M)$ is total, then M lies on a computable tree T_i such that either T_i is Ψ_i splitting or above some initial segment of M T_i has no Ψ_i splittings.

The requirements are given a priority order as follows,

$$\mathcal{R}_0 > \mathcal{R}_1 > \cdots > \mathcal{R}_i > \cdots$$

R_0 is the module of highest priority.

We define a sequence of 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 in T_s and that T_s is Ψ_s splitting or that μ_s has no Ψ_s splitting extensions on T_s. The set $M = \bigcup \mu_s$ then satisfies all the requirements.

3.2.1 Construction

Stage 0. Set $\mu_0 = \lambda$ and $T_0 = 2^{<\omega}$.

Stage s+1. We have trees $T_0 \supseteq \cdots \supseteq T_s$ and a string $\mu_s \in T_s$. We consider the following two possibilities and act as required.

If all extensions of μ_s in T_s have Ψ_{s+1} splitting extensions in T_s then define T_{s+1} to be a Ψ_{s+1} splitting subtree of T_s above μ_s and define μ_{s+1} to be the immediate left[1] successor of μ_s in T_{s+1}.

If there is some extension of μ_s in T_s which has no Ψ_{s+1} splitting extension in T_s then let μ be the least such string and let T_{s+1} be the complete subtree of T_s above μ. Define μ_{s+1} to be the immediate left successor of μ in T_{s+1}.

3.2.2 Verification

Lemmas 3.1.3, 3.1.4 and 3.1.5 are sufficient to confirm that the degree constructed is minimal, as each requirement is satisfied the first time it is visited.

This concludes the proof of Theorem 3.2.1

$\square$

[1]The choice of left here is arbitrary, we simply require that μ_{s+1} lies on T_{s+1} and properly extends μ_s.

We only asked questions recursive in $\emptyset''$ during the construction, and as $\emptyset''$ is not minimal, it must be the case that, the degree of M, $\mathbf{m} < 0''$. Given an oracle for $\emptyset''$ we are able to tell whether above every extension of a string μ there exists splittings for a given functional - see Section 3.3.3

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

During the construction of a minimal degree below $\mathbf{0}''$ in the previous section, we were able to ask directly if (for a given i) every extension of a string μ had a Ψ_i splitting. Below $\mathbf{0}'$ we are unable to do this. We are able to ask whether there exists a Ψ_i splitting above μ, but not whether a splitting exists above every extension of μ.

Theorem 3.3.1 *[Sac61]There exists a minimal degree* $\mathbf{a} \leq \mathbf{0}'$

Proof: In order to construct a minimal degree below $\mathbf{0}'$, we adapt the proof of Theorem 3.2.1 so that we build the necessary trees in stages. At each stage we have approximations to what the different trees will look like. The basic idea is that at stage s we will have an initial segment μ of M and a sequence of trees $T_0, \ldots, T_k$ for the first $k \leq s$ trees. (Where we desire the trees to have the same properties as those in the $\mathbf{0}''$ proof.) At stage s we check the approximations to splitting trees we currently have, and if possible extend our approximations. Naively, if we find a splitting for Ψ_i we guess that at every subsequent stage we will find another Ψ_i splitting, and if we don't find a Ψ_i splitting then we guess that we will never again find a Ψ_i splitting - doing this allows us to avoid the need to use a $\mathbf{0}''$ oracle. If at a stage we realise that the approximation we have to a splitting tree is incorrect (using the $\mathbf{0}'$ oracle) we may change it - details of how this works are given later. We ensure that we only change our guesses finitely many times, and that in the limit all of our trees are infinite. In this way, we are able to ensure that in the limit we construct nested trees $\{T_i\}_{i\in\omega}$ such that if $\Psi_i(M)$ is total, then either M lies on a Ψ_i splitting tree or above some initial segment of M has no Ψ_i splitting extensions - as we did for the $\mathbf{0}''$ case.

We build nested Ψ_i splitting trees, but as noted we can not simply construct T_i in one go. Initially we assume that T_i is a Ψ_i splitting tree we will be able to make M lie on. We check each T_i in stages, and let T_i^s denote the approximation to T_i at stage s. At each stage we are able to determine whether there exists (at least) one more pair of Ψ_i splitting strings in T_i^s, above our current initial segment of M. If so the construction believes that infinitely many more splittings will be found, and so we are able to continue in the belief that T_i as a splitting tree, if not then we must instead make T_i a nonsplitting tree.

As the trees need to be nested we must take care when altering the approximations we have to each splitting tree $\{T_i^s\}$. We are able to alter our guess for T_i, but if we do then T_i must remain a subtree of $T_{i'}$ for all $0 \leq i' < i$ and every $T_{i''}$ for $i'' > i$ must be abandoned, and we must build different approximations to them within the new T_i. For each version of T_i we build we are only able to change our minds once. If at some stage the $0'$ oracle does not find a splitting above μ_s for Ψ_i, then no splitting exists above μ_s. So we only alter trees which affect T_i finitely many times.

At each stage in the construction we bound the number of trees we look at, by the number of the stage we are at.

We have just one type of requirement for our construction.

$\mathcal{R}_i$: If $\Psi_i(M)$ is total, then M lies on a computable tree T_i such that either T_i is Ψ_i splitting or above some initial segment of M T_i has no Ψ_i splittings.

The requirements are given a priority order as follows,

$$\mathcal{R}_0 > \mathcal{R}_1 > \cdots > \mathcal{R}_i > \cdots$$

We use this order to ensure that the trees are nested correctly.

Each $\mathcal{R}_i$ has a functional Ψ_i and a tree T_i associated with it. We denote by T_i^s the tree associated with requirement $\mathcal{R}_i$ at stage s.

At the start ofstage $s+1$ we have an approximation to M, μ_s and a sequence of nested trees $T_0^s \supseteq \cdots \supseteq T_i^s$, for some $i \leq s$ which are the current approximations to splitting and nonsplitting trees.

3.3.1 Construction

We build a sequence of nested trees T_k^s ($k \leq\in \omega$), and a set of approximations $\{\mu_s\}_{s\in\omega}$ such that $\bigcup_s \mu_s = M$ and lies on a branch of the nested trees. The formal construction is as follows.

Stage 0: Set $\mu_0 = \lambda$, and $T_0^0 = 2^{<\omega}$.

Stage s+1: We have μ_s and some finite sequence of nested trees

$$T_0^s \supseteq \cdots \supseteq T_k^s$$

where $k \leq s$.

Find the least k' such that either

1. There are no strings in $T_{k'}^s$ which properly extend μ_s.

2. $k' = k+1$.

If $k' \leq k$ then define $T_{k'}^{s+1}$ to be the complete subtree of $T_{k'-1}^s$ above μ_s. Define $T_{k''}^{s+1} = T_{k''}^s$ for all $k'' < k'$ and make $T_{k''}^{s+1} \uparrow$ for all $k'' > k'$. Define μ_{s+1} to be a proper extension of μ_s in $T_{k'}^{s+1}$ which is not an initial segment of Ψ_s.

If $k' = k + 1$. Define μ_{s+1} to be a proper extension of μ_s in T^s_k, which is not an initial segment of Ψ_s. For all $k'' < k'$ define $T^{s+1}_{k''} = T^s_{k''}$. Define $T^{s+1}_{k'}$ to be the subtree of T^s_k above μ_{s+1}. Leave $T^{s+1}_{k''}$ undefined for all $k'' > k$.

3.3.2 Verification

In order to show that each R_i requirement is satisfied we must show that for each i our approximation to T_i is stable in the limit (i.e., each R_i is only injured finitely many times) and that each T_i is either a Ψ_i splitting tree or above some initial segment of M T_i has no Ψ_i splittings.

We show that each T_i is eventually stable, and has the require properties by induction. The base case $T_0 = 2^{<\omega}$ is trivially stable Ψ_0 splitting and contains M as a branch.

Suppose we are at stage s and that the trees $T_0, \cdots T_n$ is such that at all stages $s' > s$ when, during the construction, we search for k' the least such we find is strictly greater than n. If this is the case then for all $i \in \{0, \dots, n\}$ T^s_i contains μ_s as a branch, and by construction each $T^{s'}_i$ will contain $\mu_{s'}$ as a branch. More over if T^s_i is a Ψ_i splitting tree then by assumption T_i will be an infinite Ψ_i splitting tree, similarly if above some inital segment of μ_s T^s_i contains no further Ψ_i splittings than there will be no further Ψ_i splittings above (the same initial segment) of M in T_i.

We now argue that there exists a stage $s'' > s$ where this is also true for $T^{s''}_{n+1}$. There are two possibilities.

1. At no future stage when we search for k is the least such $k = n + 1$. In this case the induction holds, and we build T_{n+1} as a Ψ_{n+1} nonsplitting tree.

2. At stage s'' we pick $k = n + 1$. If this is the case, then we are unable to find a further splitting above some initial segment of $\mu_{s''}$. In this case we build T_{n+1} to

follow T_n. If T_n contains Ψ_{n+1} splittings above some initial segment of M we reach a contradiction.

So at all stages after s'' the value of k picked is greater than $n+1$ and so the induction holds and T_{n+1} is constructed such that above some inital segment of M it contains no further Ψ_n splittings.

This completes the proof of Theorem 3.3.1

□

3.3.3 Remarks

In the $\emptyset'$ construction we ask the question "Do there exist two strings extend μ which Ψ_i split?". This is answerable using a $\emptyset'$ oracle. We are unable to ask more though, like we do in the $\emptyset''$ construction.

In the $\emptyset''$ construction we ask "Does there exist τ extending μ such that no two strings in T_s extending τ are Ψ_i splitting?".

One way to answer this is the following search procedure: search all extensions τ of μ (ordered by length and then lexicographically) for two strings in T_s extending τ which Ψ_i split and terminate if no such exist. This is a $\emptyset'$ oracle search procedure and so computable in $\emptyset''$.

3.4 A Minimal Degree by Full Approximation

The two previous minimal degree constructions(Theorems 3.1.1 and 3.2.1), in this chapter, both used oracles but we do not always have this option. For instance suppse we wish to build a set (A of degree $\mathbf{a}$) computable in a given set (B of degree $(b) > \mathbf{a}$). If we ask questions of an oracle C of degree $\mathbf{c} > \mathbf{b}$ durign the construction we will have the $\mathbf{a} \leq \mathbf{c}$ but not necessarily that $\mathbf{a} \leq \mathbf{b}$ - as we required. An example of this is building a minimal degree below an arbitrary (given) c.e. set, as we do in Section 4.1, where using an oracle for $\emptyset'$ does not guarantee that the set constructed is below the given c.e. set.

Theorem 3.4.1 *[Yat70]It is possible to construct a minimal degree by full approximation.*

Proof[Coo72]: If we do not have access to a $0'$ oracle then we are not able to simply ask if a splitting for a certain functional exists above a given string. As for the previous constructions of minimal degrees, we build a set M and nested splitting trees so that for all i either M lies on a Ψ_i splitting tree or a Ψ_i nonsplitting tree. We are unable to ask an oracle for the splitting trees and so instead build approximations to them. This method of construction, only asks recursive questions. Our approximations, $\{\mu_s\}_{s\in\omega}$, to M are uniformly recursive. The limit of uniformly computable functions is computable in $\emptyset'$, by the limit lemma, and so $M \leq_T \emptyset'$. During our construction, where previously we were able to use the fixed listing of trees to provide the necessary nested splittings, we will approximate all of the necessary splitting trees separately.

Our tree of strategies, depicted in Figure 3.2, has one type of module:

$\mathcal{R}_i$: If $\Psi_i(M)$ is total, then M lies on a computable tree T_i such that either T_i is Ψ_i splitting or above some initial segment of M T_i has no Ψ_i splittings.

Each module has two outcomes ∞ and f - representing that $\mathcal{R}_i$ believes infinitely many Ψ_i splittings exist and that $\mathcal{R}_i$ believes that no more Ψ_i splittings exist, respectively. Each module, α, on the tree of strategies has an associated tree denoted T_α, which is initially empty, and has strings enumerated into it as we build our approximation to the appropriate splitting tree. A module's approximation to the splitting tree may be initialised, if certain conditions are met - details of this are given later.

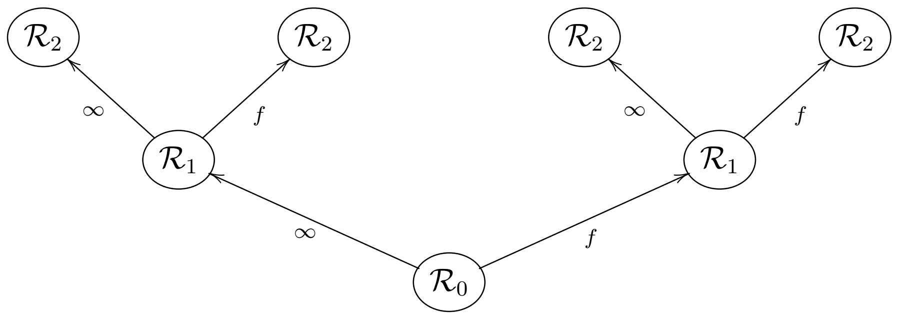

Figure 3.2: Tree of Strategies for the Minimal Degree by Full Approximation Construction

Modules may move from playing outcome ∞ to playing outcome f, and move from playing outcome f to outcome ∞. A module α changes the outcome it plays, as its belief about whether further Ψ_α splittings exist changes. The control path is the path of modules accessed during a stage s, and is denoted CP_s. The true path is the left most control path visited infinitely often, and is denoted TP.

Consider the different possible outcomes played by modules, as depicted in Figure 3.3. If both R_0 and R_1 play outcome ∞, then they both currently believe that they will find splittings for their associated functionals infinitely often. In this case the tree R_2 builds, T_2, must be within both T_1 and T_0 (i.e., $T_2 \subseteq T_1 \subseteq T_0$). If however R_1 plays outcome f, then (although R_1 will at subsequent stages continue to look for Ψ_1 splittings) all modules of lower priority on the control path do not believe that any more Ψ_1 splittings exist. In this case R_2 must build, its tree T_2 directly within T_0.

Naively, when given control a module α will search (in a complete and bounded fashion) for a pair of strings extending a (given) string which form a Ψ_α splitting. It will then enumerate these splittings into its current approximation to T_α.

In order to manage the splitting trees we are approximating, each module monitors the tree it needs to be working within. If $\alpha = \emptyset$, then $T_{\alpha^*} = 2^{<\omega}$, otherwise let α^* be the greatest $\beta \subset \alpha$ on the tree of strategies such that $\beta * f \not\subset \alpha$. Then T_{α^*} is the tree of lowest

$$\begin{tikzcd}
\mathcal{R}_2 & & & & & \mathcal{R}_2 \\
& \mathcal{R}_1 \arrow[ul, "\infty"'] & & & \mathcal{R}_1 \arrow[ur, "f"'] & \\
& & \mathcal{R}_0 \arrow[ul, "\infty"'] & & & \mathcal{R}_0 \arrow[ul, "\infty"']
\end{tikzcd}$$

Figure 3.3: Possible Outcomes on the Minimal Degree Tree of Strategies

priority which α must work within as it builds T_α. As Ψ_0 is the identity functional R_0 always plays outcome ∞ and so T_{α^*} is defined for all α.

At stage $s+1$ if μ_s extends a T^s_α leaf τ, we search for τ_0, τ_1 on $T^s_{\alpha^*}$ extending τ which form a Ψ_α splitting. If we find such strings then we enumerate them into T_α.

If α lies on the true path then either M will lie on T_a which will either be Ψ_α splitting tree or above some initial segment of M T_a will contain no further Ψ_α splittings and so we will be able to use Lemmas 3.1.3, 3.1.4 and 3.1.5 as usual.

This construction differs from the first two minimal degree constructions in that the formal construction is split into different phases. We have a phase where we define μ_s (as we require that $\{\mu_s\}$ is a computable approximation and not that $\mu_s \subset \mu_{s+1}$), and a phase where we search for and enumerate splittings.

3.4.1 Construction

At stage s by initialise T^s_α we mean make $T^{s+1}_\alpha(\rho)$ undefined on all inputs. We use $\beta <_L \alpha$ to mean that β is lexicographically less than α but $\beta \not\subset \alpha$ - intuitively we think of this as meaning that β is to the left of α on the tree of strategies.

A node α is initialised at stage s if either:

1. $s = 0$.

2. β enumerates a string into S_β and either $\beta * f \subseteq \alpha$ or $\beta <_L \alpha$.

At stage $s = 0$ all modules are initialised. At stage $s > 0$ the instructions consist of two phases.

Phase 1: Defining μ_s Perform the following finitely terminating iteration, which defines a path through the nested splitting trees.

Step 0: Define $\mu_s^* = 0$.

Step $k > 0$: Check if there exists α such that $\mu_s^* \in T_\alpha^{s-1}$, but is not a T_α^{s-1} leaf. If not define $\mu_s = \mu_s^*$. Otherwise let α the lowest priority of all such nodes. Let μ be the left successor of μ_s^* on T_α^{s-1}, redefine $\mu_s^* = \mu$ and go to the next step of the iteration.

Phase 2: Action Phase In this phase we define the control path, CP_s,inductively on $i \leq s$, and search for splittings.

Step 0: Define $CP_s \restriction_0 = \lambda\alpha_0$.

Step $i > 0$ Assume $\alpha = CP_s \restriction_{i-1}$ is defined. If $|\alpha| \geq s$ finish the stage.

If $T_\alpha^{s-1}(\emptyset) \uparrow$ set $T_\alpha^s(\emptyset) = T_{\alpha^*}^{s-1}(\rho)$ where ρ is the largest string such that $T_{\alpha^*}^{s-1}(\rho) \subset \mu_s$.

Otherwise if $\nu \subset \mu_s$ for some T_α^{s-1} leaf ν search for Ψ_α splittings above ν of length $\leq s$ on $T_{\alpha^*}^{s-1}$ and enumerate them into T_α^s. If such splittings are found define $CP_s \restriction_i = CP_s \restriction_{i-1} *\infty$. If no such splitting is found define $CP_s \restriction_i = CP_s \restriction_{i-1} *f$. Repeat *Phase 2.*

3.4.2 Verification

As is standard, all our searches are bounded, and at no point do we require the use of an oracle. During *Phase 2* of the construction we ask for the "largest ρ", this is unique so there is no ambiguity as to the string we pick.

During the verification we show the following.

1. For all n there exists a left most node of length n which is visited infinitely often and initialised finitely many times (i.e., the true path exists, and nodes on it are initialised finitely often).
2. M is total, i.e., $lim_{s\to\infty}\mu_s(n)$ exists for all n.
3. If α is on the true path and plays ∞ infinitely often then T_α is infinite and $M \leq_T \Psi_\alpha(M)$. Otherwise T_α is finite and $\Psi_\alpha(M)$ is partial or computable.

We also argue, as we have done previously, that M is incomputable using Lemma 3.1.5.

Lemma 3.4.2 *For all n there exists a left most node of length n, α_n, which is visited infinitely often and initialised finitely many times.*

Proof: We prove this by induction on n. The base case, for $\alpha_0 = \lambda$, holds trivially.

For the induction step we let $n > 0$ and assume that the result holds for all $n' < n$. The control path passes through α_{n-1} infinitely often, and every time this happens α_{n-1} must play some outcome. If α_{n-1} plays ∞ infinitely often then α_n is the left successor of α_{n-1}, otherwise it is the right successor of α_{n-1}.

α_n can only be initialised finitely often, as it is only initialised when a node $\beta <_L \alpha$ is accessed by the tree of strategies which by the hypothesis only happens finitely often, or when a node β with $\beta * f \subset \alpha$ acts, which again by induction only happens finitely often.

□

In the previous constructions we demanded that, for all s, $\mu_s \subset \mu_{s+1}$. This is not the case with this construction we simply demand that $\{\mu_s\}_{s\in\omega}$ is an approximating sequence, and so we must show that there is no value n such that $\mu_s(n) \neq \mu_{s+1}(n)$ for infinitely many s.

Lemma 3.4.3 *M is total*

Proof: We prove this by induction on the length of μ. α_0 is the module of heighest priority, and works with the identity functional. From this it follows that T_0 is infinite, and hence that for every length l there exists s such that $|\mu_s| > l$.

We consider what might happen to make μ_{s-1} and μ_s incompatible infinitely often. For this to be the case, during *Phase 1* of the construction the iterative procedure must differ in stage $s-1$ and s, let k be the least step in the iteration that this happens. If this is the case then in stage s we found an α of lowest priority such that $\mu^*_s \in T^s_\alpha$, but μ^*_s was not a T^s_α leaf (but in stage $s-1$ was, otherwise μ_s and μ_{s-1} are compatible at step k) and redefined μ^*_s to be the left successor of its previous value in T^s_α.

The existence of the TP was proven in Lemma 3.4.2, and we consider the actions of the modules on the TP now. A module α can only redfine $\mu^*_{s'}$ for $s' > s$, to be the left successor of its previous value if a tree associated with a module of higher priority is initialised. If α is on the true path this can only happen finitely often (by the previous lemma) and so there can only be finitely many stages at which μ_s and μ_{s-1} are incompatible. Hence $M = lim_s \mu_s$ is total.

□

Conceptually the bulk of the verification is in showing that the nested trees we build have the required properties - i.e., for each α, T_α is either infinite and Ψ_α splitting with M as a branch, or finite with a leaf which is extended by M and above this leaf there are no further Ψ_α splittings. Once we have shown this, Lemmas 3.1.3, 3.1.4 and 3.1.5 will give us the minimality result.

Lemma 3.4.4 *If α is on the TP and plays outcome ∞ infinitely often then T_α is infinite and $M \leq_T \Psi_\alpha(M)$, otherwise T_α is finite and $\Psi_\alpha(M)$ is partial or computable. M is not computable.*

Proof: Let α be a node on the TP, by Lemma 3.4.2 α is visited infinitely often, and only initialised finitely many times. Let s be the last stage at which α is initialised. Let s_0 be the next stage at which α is visited.

We have two possibilities to consider.

1. α only plays outcome ∞ finitely many times;
2. α plays ∞ infinitely many times.

In the first case that T_α is finite is clear. We only extend T_α if Ψ_α splittings are found extending a leaf of T_α. If this happens we play ∞. If we only play ∞ finitely many times then at some stage s' we never again play ∞. At this stage T_α is finite, and it is never again extended. *Phase 1* of the construction ensures that if T_α is finite then it contains M and that if T_α is finite then one of its leaves is extended by M.

If α plays ∞ infinitely many times then we show by induction on stage number that T_α grows unboundedly and that M lies on T_α. Let s_i be the i^{th} stage $> s_0$ in which α plays ∞.

The base of the induction holds, as at stage s_0 we define $T_\alpha^{s_0}(\emptyset) = T_{\alpha^*}^{s_0-1}(\rho) \subset \mu_{s_0-1}$. Suppose the induction holds for every $s_{i'} < s_i$ and consider the possible actions at stage s_i. During *Phase 1* μ_{s_i} is defined to be a branch of $T_\alpha^{s_i-1}$. During *Phase 2* a Ψ_α splitting is found and enumerated into $T_a^{s_i}$. So $T_\alpha^{s_i}$ is a Ψ_α splitting tree, with μ_{s_i} as a branch which is strictly larger than at the previous stage, and so the induction holds.

That is T_α is finite, or Ψ_α splitting.

Considering the two cases again, we see that in the first M lextends a leaf, τ of T_α which is finite and no Ψ_α splittings above this initial segment of M have been found. This is equivalent to saying that M lies on an infinite tree which above some initial segment of M contains no further Ψ_α splittings - simply define Ψ_α non splitting tree T'_α to be the union of T_α and the complete tree extending τ within T_{α^*}. No branches in T_{alpha^*} which extend τ form a Ψ_α splitting, as if they did they would previously have been enumerated into T_α. We are then able to use Lemma 3.1.4 to show that $\Psi_\alpha(M)$ is either partial or computable.

In the second case M lies on T_α which is an infinite Ψ_α splitting tree and so we are able to use Lemma 3.1.3 to show that $M \leq_T \Psi_\alpha(M)$.

This means that for all α either M lies on T_α which is either a computable Ψ_α splitting tree or for which above some initial segment of M contains no further Ψ_α splittings and so we are able to use Posner's trick (Lemma 3.1.5) which gives us that M is incomputable.

□

Hence we have constructed a minimal degree, below $0'$ using the full approximation method.

□

Chapter 4

Computably Enumerable Degrees and the Meet Property

In this chapter we look at the relationship between computably enumerable degrees and the meet property, leading towards our first new result - Theorem 4.2.1. We first consider how to build a minimal degree below an arbitrary c.e. degree this was first shown in [Yat70]

As per Definition 2.3.2, degree $\mathbf{a}$ satisfies the meet property if $\mathbf{a}$ is incomputable and for all $\mathbf{b} < \mathbf{a}$ there exists a non-zero degree $\mathbf{c} < \mathbf{a}$ such that $\mathbf{b} \wedge \mathbf{c} = \mathbf{0}$. In this chapter we show that an arbitrary incomputable c.e. degree satisfies the meet property. Whether this was true was first asked by Cooper and Epstein in [CE87]. In this paper a partial solution was given: if $\mathbf{a}$ is low, and $\mathbf{b} < \mathbf{a}$ is c.e. one is able to find a minimal degree $\mathbf{m} < \mathbf{a}$ for which $\mathbf{b} \wedge \mathbf{m} = \mathbf{0}$. In the paper it was further conjectured that one is unable to drop either the assumption that $\mathbf{a}$ is low or that $\mathbf{b}$ is low. In [Ish03] Ishmukhametov showed that the assumption lowness can be dropped. Further details of work in this area can be found in [Lew].

In [Sho66], Shoenfield showed that $\mathbf{0}'$ satisfies the meet property. He did this by showing that given an arbitrary degree non-zero $\mathbf{a}$ strictly below $\mathbf{0}'$ there exists a minimal degree which is below $\mathbf{0}'$ and incomparable with $\mathbf{a}$. It is this approach which we use to show that all c.e. degrees satisfy the meet property, though the implimentation is significantly more involved than in previous proofs.

First, in Section 4.1 we show that given a non-zero c.e. degree we can construct a minimal degree below it, before showing that all c.e. degrees satisfy the meet property in Section 4.2.

4.1 A Minimal Degree below a C.E. Degree

In this section we show that given an incomputable c.e. degree there exists a minimal degree below it. This was first shown by Yates in [Yat70], and in [Eps75] Epstein gave a proof of the result based on the full approximation method of constructing a minimal degree.

The proof we give is an adaptation of our full approximation construction of a minimal degree, in Section 3.4. Although the alteration is not huge, the construction of a minimal degree below a c.e. degree is essential for the main result in this chapter, Theorem 4.2.1 in Section 4.2, and so we give full details.

4.1.1 Constructions below a C.E. Degree

We give a general a method for constructing reductions or functionals (not necessarily c.e.) below a given non-zero c.e. degree. Our basic idea only ensures that the set we construct is less than the given c.e. degree and so must be combined with whatever other requirements we have for the set being constructed. In this case we combine the argument with the minimality requirements. See [Coo04] or [Nie09] for a more detailed introduction to the concept.

Given a set $A \in$ a incomputable and c.e. we may assume that we have a function f such that $f(\omega) = A$ which is $1 - 1$ and computable. We define a computable approximation $\{\mu_s\}$ of a set $M \leq_T A$. We then require that if $\forall s' > s, f(s') \geq n$ then $M \restriction_n = \mu_s \restriction_n$, i.e., if f no longer enumerates elements below n then M is fixed up to height n. Then in order to compute $M(n)$ we simply have to find an s with the given property and $M(n) = \mu_s(n)$.

Conceptually this says that any change in our approximation to M must have permission from a corresponding (sufficiently low) change in A. Given a computable enumeration $\{A_s\}_{s\in\omega}$ of A, in order to show that $M \leq A$ in stages we enumerate axioms into a functional Γ such that $\Gamma(A) = M$. If at stage s we wish to alter M at height n we may only do so if $A_{s-1} \restriction_{n'} \neq A_s \restriction_{n'}$ where n' is the use in computing M up to height n from A using Γ.

The tree of strategies is as for the minimal degree construction given in Section 3.4 and each R_i requirement is still

$\mathcal{R}_i$: If $\Psi_i(M)$ is total, then M lies on a computable tree T_i such that either T_i is Ψ_i splitting or above some initial segment of M T_i has no Ψ_i splittings.

We also have a global permitting requirement, which naively says 'a module α requires permission from A in order to enumerate a string into T_α'.

The global permitting requirement is implemented as follows. We enumerate axioms into a functional Γ such that $\Gamma(A) = M$. Axioms will be enumerated at the end of stage s for arguments $n < s$. On arguments $n < s$ we let s_0 be the maximum of n and $s_1 - 1$ for any stage $s_1 \leq s$ at which a number less than n has been enumerated into A. We let the initial segment of A_s of length $s_0 + 1$ be the use in computing $\Gamma(A_s, n)$. At the end of stage s we define $\Gamma(\sigma, n) = \mu_s(n)$, where σ is the initial segment of A_s of length $s_0 + 1$.

During a stage s a string μ is permissible if it is compatible with $\Gamma(A_{s-1})$.

As before each module α has an associated tree T_α, initially undefined and T_{α^*} is T_β where β is the module of lowest priority such that $\beta \subset \alpha$ and $\beta * f \not\subset \alpha$.

Now a module α also has an associated set, $s(\alpha)$, of pairs of strings which form Ψ_α splittings. The modules now search for a splitting, and if they find one add it into $s(\alpha)$. A splitting μ_0, μ_1 in $s(\alpha)$ may only be enumerated into T_α during stage s if both μ_0 and μ_1 are permissible.

The trees we build are nested, so if $\alpha' < \alpha$ then $T_{\alpha'} \supset T_\alpha$.

4.1.2 Construction

Given a computable enumeration $\{A_s\}_{s\in\omega}$ a set A, which is c.e. and incomputable, we define a computable approximation $\{\mu_s\}$ of a set M in stages. We also construct a functional Γ, by enumerating axioms into it, such that $\Gamma(A) = M$. To accommodate the the global permitting requirement each stage of the construction consists of 4 phases. We define μ_s and search for splittings, as in Section 3.4, and have two extra phases - one which checks if splittings are permissible and if so enumerates them into trees, and one which enumerates axioms into Γ. A module α is active if it has been visited subsequent to its last initialisation

By initialise α we mean make $T_\alpha(\rho) \uparrow$ for all ρ and empty $s(\alpha)$ of all splittings it is seeking permission for. α is initialised at stage s if

1. $s = 0$.

2. β enumerates a string into S_β and either $\beta * f \subseteq \alpha$ or $\beta <_L \alpha$.

At stage $s = 0$ initialise all nodes. At stage $s > 0$ the construction acts as follows

Phase 1: Tree Enumeration For each active module α, in order of priority, search $s(\alpha)$ for the first Ψ_α splitting which is permissible. If such a splitting exists enumerate it into T^s_α and empty $s(\alpha)$.

Phase 2: Defining μ_s Perform the following finitely terminating iteration, which defines a path through the nested splitting trees.

Step 0: Define $\mu^*_s = 0$.

Step $k > 0$: Check if there exists α, with $|\alpha| < s$, such that $\mu^*_s \in T^s_\alpha$, but is not a T^s_α leaf. If not define $\mu_s = \mu^*_s$. If so let α the lowest priority of all such nodes. Let μ be the left successor of μ^*_s on T^s_α, redefine $\mu^*_s = \mu$ and go to the next step.

Phase 3: Splitting Phase In this phase we define the control path, CP_s,inductively on $i \leq s$, and search for splittings.

Step 0: Define $CP_s \upharpoonright_0 = \lambda\alpha_0$.

Step $i > 0$: If $T^s_\alpha(\emptyset) \uparrow$ set $T^s_\alpha(\emptyset) = T^s_{\alpha^*}(\rho)$ where ρ is the largest string such that $T^s_{\alpha^*}(\rho) \subset \mu_s$.

Otherwise if $\eta \subset \mu_s$ for some T^s_α leaf η search for Ψ_α splittings above η of length $\leq s$ on T_{α^*} and enumerate them into $s(\alpha)$.

If a splitting has been enumerated into T_α since we last visited α define $CP_s \upharpoonright_{i+1} = CP_s \upharpoonright_i * \infty$, otherwise define $CP_s \upharpoonright_{i+1} = CP_s \upharpoonright_i * f$. Repeat *Phase 3.*

Phase 4: Enumerating Γ On arguments $n < s$ for which $\mu_s(n) \downarrow$ let s_0 be the maximum of n and $s_1 - 1$ for any stage $s_1 \leq s$ at which a number less than n has been enumerated into A. Let σ be the initial segment of A_s of length $s_0 + 1$ and define $\Gamma(\sigma, n) = \mu_s(n)$.

4.1.3 Verification

No c.e. degree is minimal and so the A permitting requirements, which give $M \leq_T A$, are therefor sufficient to show that $M \neq_T A$, and so $M <_T A$.

We must argue that at every stage μ_s is permissible, that M is total and that $\Gamma(A) = M$. We must also show that any delays caused by waiting for splittings to become permissible do not affect the minimality arguments. If we can show these properties hold, then we may argue that M is minimal using Lemmas 3.1.3, 3.1.4 and 3.1.5.

Lemma 4.1.1 *For all stages s, μ_s is permissible, $lim_s \mu_s = M$ and $\Gamma(A) = M$*

Proof: By assumption Ψ_0 is the identity functional, and so T_0 is infinite. From this it follows that given a length l, there exists an s such that $|\mu_s| > l$. The use of Γ on argument n is also bounded (by *Phase 4*), and so we see that for all stages s if μ_s is permissible (i.e., compatible with $\Gamma(A_s)$) then M is total and $\Gamma(A) = M$.

We prove that at each stage s μ_s is permissible by induction on s, considering what could occur to make μ_{s-1} permissible but μ_s not permissible.

At stage $s = 0$, $\mu_0 = 0$ and $\Gamma = \emptyset$ and so the base case holds. Assume that the induction holds for all $s' \leq s - 1$. If μ_s and μ_{s-1} are compatible, then the induction holds.

μ_s is defined during *Phase 2* of stage s and so if μ_s and μ_{s-1} are incompatible then the action taken by the iterative processes described in *Phase 2* must differ in stage s and stage $s - 1$. Let k be the least step in the process at which the iterations diverge.

At each step of the iterative process either we find α of lowest priority such that $\mu_s^* \in T_\alpha$, but is not a T_α leaf and redefine μ_s^* to be the left successor of its previous value (on T_α) or the iterative process terminates. If μ_s and μ_{s-1} are incompatible, then at step k of the iteration in stage $s - 1$, μ_s^* was a leaf of T_α, but in stage s it is not i.e., a new Ψ_α splitting, μ_0, μ_1 say, was permitted to be enumerated into T_α in *Phase 1* of stage s. Both halves of the splitting enumerated into T_α are permissible (as described in *Phase 1*). In stage s the left half of the splitting is chosen to be μ_s^*. The iterative process ends here, as no module of lower priority than α has enumerated any splittings extending μ_0 within the newly redefined T_α, and so μ_s is permissible.

□

Lemma 4.1.2 *There exists a TP, and any node on the TP is initialised only finitely often.*

is proven as its counterpart in 3.4.2.

We now show is that the permitting requirements do not affect the minimality requirements.

Lemma 4.1.3 *If a module α, on the true path, has to wait to enumerate a splitting into T_α or after some finite stage s is never permitted to enumerate a splitting into T_α (despite finding them), then M still either lies on a Ψ_α splitting or Ψ_α nonsplitting tree.*

Proof: We consider three possibilities.

1. If after some finite stage s α never again finds a Ψ_α splitting extending a leaf of T_α, then the permitting requirements do not have any effect on the minimality requirements, and α plays outcome f.

2. If at some stage s α finds a Ψ_α splitting but has to wait until stage $s' > s$ for permission to enumerate it, then the wait is finite. This means that at some finite stage α receives persmission to enumerate the splitting into T_α during *Phase 1* of the construction. Infinitely often α may find a Ψ_α splitting extending a leaf on T_α^s, but have to wait to enumerate it. At stage s', $T_\alpha^{s_i'}$ is a Ψ_α splitting tree, with the new Ψ_α splitting enumerated into it. If this happens infinitely often then $lim_{s\to\infty}T_\alpha^s$ is a Ψ_α splitting tree, with M as a branch.

3. If α finds infinitely many splittings, but never gets permission from A to enumerate them then we are able to argue that A is computable. The idea is that we can use the stages at which the Ψ_α splittings are found to give a bound on the last stage at which A_s may change below a certain height. To see this note that if α finds a splitting during stage s, but never receives permission to enumerate it then either μ_0 or μ_1 (or both) never becomes permissible. Let μ be the longer of μ_0 and μ_1 which does not receive permission. Let μ' be the longest initial segment of μ which is permissible (this exists as we are searching for an extension of some string $\eta \subset \mu_s$ which is permissible), and let $n = |\mu'|$. At stage s the *use* in computing μ_s from A is defined (as in *Phase 4*) to be $s_0 + 1$, and for all stages $s' > s$ $A_{s'} \upharpoonright_{s_0+1} = A_s \upharpoonright_{s_0+1}$. The *use* of Γ in computing $\mu_s(n)$ increases as n increases, and so when we find further splittings at subsequent stages, the initial segment of A which is fixed grows. To compute $A(x)$ we simply run the computation until we reach a (finite) stage s where find a Ψ_α splitting, and the *use* in computing M via Γ is $\geq x$. Then $A(x) = A_s(x)$.

□

This shows us that the permitting requirement does not affect the ability of the minimality requirements to act, and so we prove

Lemma 4.1.4 *M is minimal.*

as we do its counterpart in Section 3.4.2.

4.2 C.E. Degrees and the Meet Property

In this section we prove the main result of this chapter:

Theorem 4.2.1 *Given any non-zero c.e. degree* $\mathbf{a}$ *and any degree* $\mathbf{b} < \mathbf{a}$*, there is a minimal degree* $\mathbf{m} < \mathbf{a}$ *such that* $\mathbf{m} \nleq \mathbf{b}$.

From this the following corollary is immediate

Corollary 4.2.2 *Every c.e. degree satisfies the meet property.*

We are given an incomputable c.e. set A, with computable enumeration $\{A_s\}_{s\in\omega}$. We are also given a set $B <_T A$ (which is necessarily Δ^0_2, but need not be c.e.) and a Turing functional Φ such that $\Phi(A) = B$. By speeding up the enumerations of Φ and A as necessary we get a computable approximation $\{B_s\}_{s\in\omega}$ to B. Each B_s is a finite binary string of length s, such that $B_s \subseteq \Phi(A_s)[s]$. Although each B_s is a finite binary string, we consider A_s to be an infinite binary string, which is still the characteristic function of a finite set.

We then construct a set M which is of minimal degree, and is computable from A, but $M \nleq_T B$. We construct M in stages and let μ_s be our approximation to M at stage s. Then $lim_s \mu_s = M$ and $\{\mu_s\}_{s\in\omega}$ forms a computable approximation to M.

Tree of Strategies

As for our previous constructions, we place the requirements on a tree of strategies. This construction has two distinct types of requirements placed on the tree, as well as a global permitting requirement. We ensure that M is minimal building nested splitting and nonsplitting trees, as we have done previously. These requirements are labelled $\mathcal{M}_e$. We also have requirements which aim to show that $M \nleq_T B$. These requirements are labelled $\mathcal{P}_e$, and broadly each looks to show $M \neq \Psi_e(B)$.

As depicted in Figure 4.1 a node on the tree of length $2e$ is assigned the requirement $\mathcal{M}_e$. As before this node has two outcomes $\infty <_L f$, where ∞ means that splittings are found

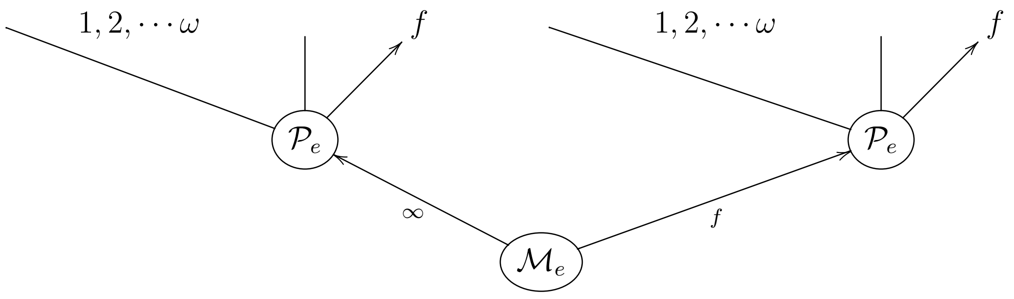

Figure 4.1: Tree of Strategies for the C.E. Meet Construction

above infinitely many initial segments of M and f means that this is not true. A node on the tree of length $2e + 1$ is assigned the requirement $\mathcal{P}_e$. Each $\mathcal{P}_e$ requirement has infinitely many outcomes labelled $0 <_L 1 <_L 2 <_L \cdots <_L f$. The set of outcomes for $\mathcal{P}_e$ has order type $\omega + 1$, with a single distinguished rightmost outcome. f means that either there is some argument m for which $\Psi_e(B_s, m) \downarrow$ for only finitely many s, or else successful diagonalisation has been achieved. Naively each of the other outcomes is a guess as to the least m for which there are infinitely many stages at which $\Psi_e(B_s, m) \downarrow$ with different *uses* (i.e., the observed *uses* are unbounded).

During the construction we use the α, β and γ to denote nodes on the tree of strategies. σ is used to denote potential initial segments of A, and τ is used to denote potential initial segments of B. We use both μ and η to denote potential initial segments of M. We let ρ range more generally over binary strings. If $\rho \neq \emptyset$ then $\rho^\dagger$ is the binary string which is the same length as ρ but differs only in the final bit. ρ^- is as before. If α is $\mathcal{M}_e$ (or $\mathcal{P}_e$) requirement on the tree of strategies then as before we may denote Ψ_e as Ψ_α.

4.2.1 Outline Proof

This construction is more involved than those given previously, and so we give a broad outline of how the different requirements work before giving a formal construction.

Permitting

To ensure that $M \leq_T A$, we use the method discussed in Section 4.1.1. We must adapt this method, as in order to attempt diagonalisation we are now also concerned with the movement of B. The process is the same, but the height of A we look at alters. We enumerate a functional Γ, such that $\Gamma(A) = M$. Axioms are enumerated at the end of each stage s, for arguments $n < s$. We choose the use for argument $n < s$ as follows. Let s_0 be the maximum of n and $s_1 - 1$ for any stage $s_1 \leq s$ at which a number less than n has been enumerated into A. Let σ be the shortest initial segment of A_s such that $B_s \restriction_{s_0+1} \subseteq \Phi(\sigma)$ - Φ is the functional we use to compute B from A. Such a σ exists by our conventions regarding B_s. At the end of stage s we define $\Gamma(\sigma, n) = \mu_s(n)$.

During a stage s, η is permissible if it is compatible with $\Gamma(A_{s-1})$.

$\mathcal{M}_e$ Requirements

To ensure that M is minimal we reemploy the notion of nested splitting trees used previously. For each Ψ_α we aim to build an infinite Ψ_α splitting or nonsplitting tree with M as a branch. The basic process is as for the full approximation construction given in Section 3.4. We use the same notation conventions as in Section 4.1, but reiterate them here.

Each $\mathcal{M}_e$ requirement α has an associated tree T_α, initially undefined. T_{α^*} is T_β where β is the $\mathcal{M}$ requirement of lowest priority such that $\beta \subset \alpha$ and $\beta * f \not\subset \alpha$, unless $\alpha = \emptyset$ in which case $T_{\alpha^*} = 2^{<\omega}$. T^s_α is α's approximation to either a Ψ_α splitting or nonsplitting tree at the end of stage s.

Each $\mathcal{M}_e$ requirement α also has an associated set, $s(\alpha)$, of pairs of strings which form Ψ_α splittings. Each requirment searches for a Ψ_α splitting extending a leaf of T_α, and if they find one add it into $s(\alpha)$. A splitting μ_0, μ_1 in $s(\alpha)$ may only be enumerated into T_α

during stage s if both μ_0 and μ_1 are permissible.

The trees we build are nested, so if $\alpha' < \alpha$ then $T_{\alpha'} \supset T_\alpha$.

$\mathcal{P}_e$ Requirements

To satisfy a $\mathcal{P}_e$ node, α, we must ensure that if Ψ_e is total then there exists some argument n for which $M(n) \neq \Psi_\alpha(B, n)$. As $B \leq_T A$, any change in the approximation to B at stage s must be witnessed by a sufficiently low change in A at stage s, similarly for M. By carefully monitoring when the A changes occur, and checking if B changes occur, we are able to decided what M changes we wish to make.

Naively we implement this idea as follows. Monitor $\Psi_\alpha(B) \restriction_n$ for a fixed n. If at some stage s, $\Psi_\alpha(B) \restriction_n$ agrees with $\mu_s \restriction_n$ then a suitably low A change would mean that we could change our approximation to M below n. If it remains the case that $B_s \subset B$, then we will have successfully diagonalised.

While we wait for an A change we map the initial segment of B involved in the computation of Ψ_α to the initial segment of A we are hoping to see a change in, and start working again with a larger n. As $A \not\leq_T B$, we must either find that we get some A-permission to diagonalise (i.e., we see an A-change with no corresponding B-change), or there is some n such that $M \restriction_n$ is not an initial segment of $\Psi_\alpha(B)$.

Competing Requirements

Although the idea of using diagonalisation and permitting to ensure that M is below A and incomparable with B seems intuitive, we must coordinate the diagonalisation with the minimality requirements and it is here that the difficulties lie. Each $\mathcal{P}_e$ node α builds a functional Φ_α, with which it threatens to compute A from B. It does this using modules. Each node α maintains infinitely many modules $M^0_\alpha, M^1_\alpha, \cdots$. The module M^i_α

is responsible for enumerating axioms into $\Phi_\alpha(i)$. The node α works within the tree T_{α^*}, where T_{α^*} is as previously defined.

We build $\Phi_\alpha(i)$ as a c.e. set of strings and ensure that after i enters A (if it ever does) no more strings are enumerated into $\Phi_\alpha(i)$. A is c.e. so if i enters A, it is never removed. Then to compute $\Phi_\alpha^X(i)$ we run the enumeration of $\Phi_\alpha(i)$ until either a string τ is found such that $\tau \subset X$, or else i enters A. In the former case we output 0, in the latter we output 1.

While $i \notin A$, the module M_α^i waits until it sees $\eta \subset \Psi_e(\tau)$ for some $\tau \subseteq B_s$ and $\eta \subseteq \mu_s$ such that $\eta = T_{\alpha^*}(\rho)$ for some ρ which is specific to this module. Then it enumerates τ into $\Phi_\alpha(i)$ as well as issuing the *demand* $(\tau, i, \eta_0, \eta_1)$, where $\eta_0 = T_{\alpha^*}(\rho^-)$ and $\eta_1 = T_{\alpha^*}(\rho^\dagger)$. This demand is read "if $\tau \subset B$ and $i \in A$, then $\eta_0 \subset M \Rightarrow \eta_1 \subset M$".

When a demand is acted upon and plays a role in the definition of μ_s at stage s, we say that it is *implemented* at stage s. The precise definition of implementation is given during the formal construction.

Having demands of this form might seem unnecessarily complicated. But when we consider possible alternatives problems occur, due to the necessary A-permissions - specifically the need for modules to the left of the true path to require action during a stage. For instance if we were to issue the demand "if $\tau \subset B$ and $i \in A$, then $\eta_1 \subset M$" (rather than "if $\tau \subset B$ and $i \in A$, then $\eta_0 \subset M \Rightarrow \eta_1 \subset M$") then if $i \in A$ we have permission to change our mind about whether $\eta_i \subseteq M$ as our information about whether $\tau \subseteq B$ changes, as long as μ_0 is an initial segment of the approximations we have to M.

4.2.2 Construction

Given a computable enumeration $\{A_s\}_{s\in\omega}$ of A, c.e. and incomputable, and a set $B <_T A$ we construct a set M in stages μ_s such that $lim_s \mu_s = M$. We are also given a Turing functional Φ, such that $B = \Phi(A)$ and by speeding up the enumeration of Φ and A

we have an approximation $\{B_s\}_{s\in\omega}$. As well as M we build a functional Γ such that $\Gamma(A) = M$.

We divide each stage of the construction into four phases.

If α is an $\mathcal{M}_e$ requirement then by initialise α we mean make $T_\alpha(\rho) \uparrow$ for all ρ and set $s(\alpha) = \emptyset$. If α is a $\mathcal{P}_e$ requirement then by intitialise α we mean discard all axioms enumerated for Φ_e, and all demands issued by modules maintained by α. We also discard all *recorded computations* (as defined in *Phase 4*) for α.

All nodes α are assigned a number (z_α - larger than any previously used) the first time they are visited after an initialisation. When any node α is initialised z_α is made to be undefined.

A node α is initialised at stage s as soon as one of the following conditions is met - irrespective of whether it is an $\mathcal{M}_e$ node or a $\mathcal{P}_e$ node.

1. $s = 0$.
2. A node strictly to the left of α is visited.
3. β enumerates strings into S_β at stage s and either $\beta * f \subseteq \alpha$ or $\beta <_L \alpha$.
4. A demand issued by a module M^j_β such that either $\beta <_L \alpha$ or $\beta * i \subseteq \alpha$ for some $i < j$ is *implemented* (as defined in *Phase 2*) at stage s, but that demand was not implemented at stage $s - 1$ or vice-versa, the demand was implemented at stage $s - 1$ but is not implemented at stage s.

A module α is active if it has been visited subsequent to its last initialisation. T_α is active if α is active.

Stage 0**.**

All nodes are initialised.

Stage $s > 0$**.**

At stage $s > 0$ the construction is divided into four phases as follows.

Phase 1: Tree Enumeration For each active module, α, assigned a minimality requirement, in order of priority, search $s(\alpha)$ for the earliest enumerated splitting which is permissible. If such a splitting exists enumerate it into T^s_α and empty $s(\alpha)$.

Phase 2: Defining μ_s Perform the following finitely terminating iteration, which defines a path through the nested splitting trees. The process takes into account issued demands (where they exist) and takes the left path otherwise. As we proceed, we also enumerate pairs of the form (μ, B) in order to keep track of the priority with which we have implemented demands.

Step 0: Define $\mu^*_s = \lambda$.

Step $k > 0$: Check if there exists a demand issued by some M^i_β of the form $(\tau, i, \eta_0, \eta_1)$ such that $\tau \subseteq B_s$, $i \in A_s$, $\eta_0 \subset \mu^*_s$ and such that we have not already enumerated any pair (μ, γ) during the iteration at stage s with $\mu \supset \eta_0$ and γ of higher priority than β. If so, choose that for which η_0 is shortest, declare that this demand is implemented at stage s, redefine $\mu^*_s = \eta_1$, enumerate the pair (η_1, β) and go to the next step. Otherwise check to see if there exists α such that $\mu^*_s \in T_\alpha$ but μ^*_s is not a T_α leaf. If not then define $\mu_s = \mu^*_s$ and terminate the process, otherwise let α be the lowest priority of all such nodes. Let μ be the left successor of μ^*_s on T_α, and then redefine μ^*_s to be μ and go to the next step.

Note: Implemented demands may subsequently be injured by another demand of higher priority i.e., for the implemented demand $(\tau, i, \eta_0, \eta_1)$ it may not always be the case that $\eta_1 \subseteq \mu_s$.

Phase 3: Visiting Phase In this phase we define the control path, CP_s, inductively on $i \leq s$, and (depending on the requirement type) search for splittings or look to diagonalise.

Step 0: Define $CP_s \restriction_0 = \lambda = \alpha_0$. Define $T_{\alpha_0} = \emptyset$.

Step i: Assume $\alpha = CP_s \restriction_i$ is defined. If $|\alpha| \geq s$ finish the phase. Otherwise if z_α is not already defined choose it to be an odd number larger than previously used. Then consider the following two cases:

1. α is assigned $\mathcal{M}_e$. If $T_\alpha(\emptyset) \uparrow$ and $T_{\alpha^*}(\rho) \subset \mu_s$ then set $T_\alpha(\emptyset) = T_{\alpha^*}(\rho)$ where $|\rho| = z_\alpha$ If no such ρ exists leave $T_\alpha(\emptyset)$ undefined.

   Otherwise (i.e., T_α is non-empty) if $\eta \subset \mu_s$ for some T_α-leaf η search for Ψ_α splittings above η of length less than s, on T_{α^*}, and enumerate them into s_α.

   If a splitting has been enumerated into T_α since we last visited α define $CP_s \restriction_{i+1} = CP_s \restriction_i *\infty$, otherwise define $CP_s \restriction_{i+1} = CP_s \restriction_i *f$. Repeat *Phase 3*.

2. α is assigned $\mathcal{P}_e$. Determine the least $j < s$ such that M_α^j *requires attention.* M_α^j *requires attention* if there exists $\mu \subset \mu_s$ such that $\mu = T_{\alpha^*}(\rho)$ for ρ of length $p_j^{z_a}$ (i.e., the j^{th} prime raised to the z_α^{th} powers) and (for some shortest string$\tau \subseteq B_s$) $\mu = \Psi_e(\tau)[s]$, but M_α^j has not yet *recorded the computation* (as defined in the next paragraph).

   If no M_α^j requires attention then α performs no action and sets $CP_s \restriction_{i+1} = CP_s \restriction_i *f$. Otherwise let j be the least such that M_α^j requires attention and declare $\Psi_e(\tau)$ to be a *recorded computation.* If $j \notin A_s$ then issue the demand $(\tau, j, \eta_0, \eta_1)$, where ρ is as above, $\eta_0 = T_{\alpha^*}(\rho^-)$ and $\eta_1 = T_{\alpha^*}(\rho^\dagger)$. Enumerate τ into $\Phi_\alpha(j)$, set $CP_s \restriction_{i+1} = CP_s \restriction_i *j$ and repeat *Phase 3.*

Phase 4: Enumerating Γ On arguments $n < s$ for which $\mu_s(n) \downarrow$ let s_0 be the maximum of n and $s_1 - 1$ for any stage $s_1 \leq s$ at which a number less than n has been enumerated into A. Let σ be the shortest inital segment of A_s such that $B_s \restriction_{(s_0+1)} \subseteq \Gamma(\sigma)$ and define $\Gamma(\sigma, n) = \mu_s(n)$.

4.2.3 Verification

In the verification we must show the following:

1. At every stage μ_s is permissible, M is total, i.e., $lim_{s\to\infty}\mu_s(n)$ exists for all n and $\Gamma(A) = M$.
2. Each node on the true path is only initialised finitely many times, each $\mathcal{P}_e$ node is satisfied, for all $\mathcal{M}_e$ requirements, α, if $\Psi_e(M)$ is total, then M lies on a computable tree T_α such that either T_α is Ψ_i splitting or above some initial segment of M T_i has no Ψ_i splittings.

If we can show 2 then we can argue that M is minimal using Lemmas 3.1.3, 3.1.4 and 3.1.5 as we have done previously.

First we verify that all the instructions are well defined. The only place where any ambiguity occurs in the construction, during a stage s, is during *Phase 2* of the construction. During *Phase 2*, at step k we are required to select the appropriate demand $(\tau, i, \eta_0, \eta_1)$ for which η_0 is shortest. We must verify that there is a unique such demand. Once we have shown that there is a least such demand it will be clear that the instructions for each stage (particularly during *Phase 2*) are finite as:

1. If the demand $(\tau, i, \eta_0, \eta_1)$ is implemented during step k of *phase 2* of stage s, and $(\tau', j, \eta_2, \eta_3)$ is implemented at step $k' > k$ of the same stage then η_2 properly extends η_0.
2. At each stage only finitely many demands are issued, and only finitely many strings are enumerated into trees.

So we must ensure that at any point of the construction if $(\tau, i, \eta_0, \eta_1)$ and $(\tau', j, \eta_2, \eta_3)$ are both issued demands which have not been discarded due to some initialisation, then

$\eta_0 = \eta_2$ implies $i = j$ and that both demands where implemented by the same module, namely M^i_α.

As z_α was chosen to be large whenever a node was visited for the first time, subsequent to initialisation, we have that when $\mu \in T_\alpha \cap T_{\alpha'}$, we must also have that $\alpha * \infty \subset \alpha'$ or vice-versa, i.e., $\alpha' * \infty \subset \alpha$. That is when a string belongs to two valid trees it must be the case that one of these trees is purposely built as a subtree of the other. Considering the following:

1. For α of even length, strings in T_α are of odd length in T_{α^*}.
2. If M^i_α issues a demand $(\tau, i, \eta_0, \eta_1)$ then η_0 is of even level in T_{α^*}.
3. If two nodes α_1 and α_2 are such that they are both of odd length, both valid and $T_{\alpha_1^*} = T_{\alpha_2^*}$ then, as $z_{\alpha_1} \neq z_{\alpha_2}$ it follows that for any two demands $(\tau, i, \eta_0, \eta_1)$ and $(\tau', j, \eta_2, \eta_3)$ issued by modules $M^i_{\alpha_1}$ and $M^j_{\alpha_2}$ respectively, we must have $\eta_0 \neq \eta_2$.

we see that no two demands contradict each other, and so the construction is well defined and at the instructions are at each stage finite.

With this done, we can begin the verification proper.

Lemma 4.2.3 *At every stage s, μ_s is permissible, $lim_s \mu_s = M$ is total and $\Gamma(A) = M$*

.

Proof: A is incomputable, and as α_0 looks for splittings for the identity functional, we see that $T_\emptyset$ is infinite. It follows that for any given length l, there exists some s such that $|\mu_s| > l$. The use of Γ on argument n is bounded (clearly, by the construction), that $lim_s \mu_s = M$ and $\Gamma(A) = M$ follows from μ_s being permissible for every stage s.

We prove that μ_s is permissible at every stage, s, by induction on s. The base case is trivial. Suppose that μ_s is incompatible with μ_{s-1} and consider the iterations which take place during *Phase 2* of the construction during stage s and $s-1$. These must diverge at some point, otherwise μ_s and μ_{s-1} are compatible and the induction follows directly. We let k be the least step at which the iterations for stage s and $s-1$ diverge and consider why this might have happened. At each step of the iteration either a demand is implemented; we find α of lowest priority such that $\mu_s^* \in T_\alpha$ but is not a T_α leaf and redefine μ_s^* to be the left successor of its previous value on T_α; or the iteration is terminated. With this in mind we consider the following three possibilities, as to why divergence occurred at step k of the iteration.

1. During step k at stage $s-1$ a demand $(\tau, i, \eta_0, \eta_1)$ is implemented, and during step k of stage s no demand $(\tau', j, \eta_2, \eta_3)$ such that $\eta_2 \subset \eta_0$ is implemented.

   If this is the case then i was enumerated into A at a stage $> |\tau|$, and as $|\eta_0| > i$, any $\sigma \subset A_{s-1}$ such that $\eta_0 \subseteq \Gamma(\sigma)$ at the end of stage $s-1$ is sufficiently long that $\tau \subseteq \Phi(\sigma)$. As the demand $(\tau, i, \eta_0, \eta_1)$ is not implemented at stage s we know that $\tau \not\subset B_s$ and hence any extension of η_0 is permissible.

2. The reverse of case 1 happens, i.e., during step k at stage s a demand $(\tau, i, \eta_0, \eta_1)$ is implemented, and during step k of stage $s-1$ no demand $(\tau', j, \eta_2, \eta_3)$ such that $\eta_2 \subset \eta_0$ is implemented.

   This case can be divided down further. As the demand $(\tau, i, \eta_0, \eta_1)$ was not implemented at stage $s-1$ it is possible that either (a) i was enumerated into A at stage s, in which case any extension of η_0 is permissible of (b) we must have that $\tau \not\subset B_{s-1}$. The induction follows for (b), as any extension of η_0 is permissible. For (a) we argue similarly to case 1. i was enumerated into A at a stage $s' > |\tau|$, as η_0 is of length $> i$, any $\sigma \subset A_{s-1}$ such that $\eta \subseteq \Gamma(\sigma)$ at the end of stage $s-1$ is sufficiently long that $\tau' \subseteq \Phi(\sigma)$, where τ' is the initial segment of B_{s-1} of length s'. It follows that, as for case 1, any extension of η_0 is permissible.

3. Cases 1 and 2 don't hold, but the iteration does not terminate, i.e., we find α of

lowest priority such that $\mu_s^* \in T_\alpha$ but is not a T_α leaf and redefine μ_s^* to be the left successor of its previous value on T_α.

In this case $\mu := \mu_s^*$ (before its redefinition at step k) was a leaf of T_α prior to stage s. We let μ' be the longest string which is the longest initial segment of both successors of μ in T_α, and also of μ_{s-1}, and consider two further possibilities.

If $\mu' \subset \mu_s$ then μ_s is permissible.

Otherwise there must be some demand $(\tau, i, \eta_0, \eta_1)$ such that $\eta_0 \subset \mu'$, and which is 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 *strongly* diverged at step k, as the same demand was implemented anyway at the next step. We consider instead the least step at which the two iterations *strongly* disagree. In this case a demand was implemented at step $k+1$ of stage s, which was not implemented at step $k+1$ of stage $s-1$. The two cases that this provides us with are identical to (1) and (2), with k replaced by $k+1$.

□

Lemma 4.2.4 *For all n, there exists a leftmost node of length n which is visited infinitely often, α_n, say. This node satisfies the following:*

1. *α_n is initialised only finitely many times*

2. *If α_n is of length $2e+1$ then it ensures that $\mathcal{P}_e$ is satisfied. Either α_n has outcome f at all but finitely many stages at which it is visited, or else there exists some least m such that α_n has outcome m at infinitely many stages.*

3. *If α_n is of length $2e$ and has outcome come ∞ at infinitely many stages then T_α is infinite and $M \leq_T \Psi_e^M$. Otherwise T_α is finite and Ψ_e^M is partial or computable.*

Proof: We prove this by induction on n. The result fot $n = 0$ follows trivially from our assumptions about Ψ_0.

Suppose that $n > 0$ and that the induction holds for all $n' < n$. Then (2) of the induction hypothesis implies that α_n exists, i.e., there exists a leftmost node of length n which is visited at infinitely many stages, and also that there are only finitely many stages at which nodes strictly to the left of α_n are visited. Also, (3) of the induction hypothesis implies that for β of even length with $\beta * f \subset \alpha_n$, T_β is finite. This $\beta <_L \alpha_n$ are only visited finitely many times, and so cane only enumerate finitely many splittings into their lists. We conclude that α_n satisfies any of the the conditions for initialisation (1), (2) or (3) at only finitely many stages. We are left to deal with initialisation condition (4).

Those modules M^j_β such thate eitehr $\beta <_l \alpha_n$ or $\beta * i \subseteq \alpha_n$ for $j < i$ can only enumerate finitely many demands. Consider one such demand $\tau, j, \eta_0, \eta_1)$,issued by $M^j_{\beta_0}$ say. If $\tau \not\subset B, j \notin A$ or $\eta_0 \not\subset M$ then at all sufficiently late stages this dmeand is not implemented (if implemented at stage s then $\eta_0 \subset \mu_s$). On the other hand, for any stage s at which $\tau \subseteq B_s$, $j \in A_s$ and $\eta_0 \subseteq \mu_s$, the only way in which the demand could fail to be implemented (we are not concerned with injury) would be the implementation of a demand of higher priority $(\tau', k, \eta_2, \eta_3)$, such that $\eta_2 \subset \eta_0$ and $\eta_3 \supset \eta_0$. When two distinct trees T_α and $T_{\alpha'}$ are nested (i.e., when it is not the case that $\alpha * \infty \subset \alpha'$ or $\alpha' * \infty \subset \alpha$), initialisation means that all of the strings in one of the trees are of strictly greater length than all strings in the other. Let β_1 be the node which issued the demand $(\tau', k, \eta_2, \eta_3)$. Since $\eta_2 \subset \eta_0$ and $\eta_3 \supset \eta_0$, it must be that $T_{\beta_0^*}$ and $T_{\beta_1^*}$ are nested. Since β_1 is of higher priority, $T_{\beta_0^*}$ must either be equal to $T_{\beta_1^*}$ or built as a subtree of it. This contradicts the condition $\eta_0 \subset \eta_0$ and $\eta_3 \supset \eta_0$, given that η_3 is a successor of η_2 in $T_{\beta_1^*}$. So for each member of this finite set of demands there is either a stage after which they are always implemented, or else a stage after which they are never implemented. Thus α_n is initialised only finitely many times.

Now suppose that α_n is of length $2e + 1$. We wish to show that any demand $(\tau, i, \eta_0, \eta_1)$ issued by α_n subsequent to its final initialisation is *met*, i.e., if $\tau \subset \beta$, $i \in A$ and $\eta_0 \subset M$, then $\eta_1 \subset M$. (That is, under these conditions there is a stage after which the demand is always implemented and not injured). The argument above, that α_n only satisfies (4) of the initialisation conditions at finitely many stages, suffices to show that at any stage at which $\tau \subseteq B_s$, $i \in A_s$ and $\eta_0 \subset M$, the demand is implemented. In order for the demand to be injured we would thne have to implement another demand $(\tau^{|prime}, j, \eta_2, \eta 3)$ of higher priority, at a later step of the iteration for *phase 2* of that stage, for which $\eta_0 \subset \eta_2 \subset \eta_1$. Initialisation means that β which issued this demand, cannot satisy $\beta <_L \alpha_n$ (since α_n chooses z_{α_n} large). In fact $\beta * k \subset \alpha_n$ for some $k \in \omega$ with $k \leq j$. The finite length of η_1 also means that there are only finitely many possible values for j, and in order for any such demand to be implemented, it must be the case that the dmeand is issued at a stage prior to one at which j is enumerated into A. Thus there can only be issued finitely many demands $(\tau', j, \eta_2, \eta_3)$ of the correct form to cause injury to the demand $(\tau, i, \eta_0, \eta_1)$. The fact that the injuring demand is issued by M^j_β and $\beta * k \subset \alpha_n$ for some $k \in \omega$ with $k \leq j$, means that there is a stage s such that for all $s' \geq s$, $\tau' \not\subset B_s$ and the potentially injuring demand is not implemented.

Now if α_n has outcome f at all sufficiently large stages at which it is visited, or else there exists some least m such that α_n has outcome m at infinitely many stages, then it is clear that $\mathcal{P}_e$ is satisfied. So suppose that this does not hold. Then $\Psi_e(B) = M$. For each $i \notin A$, there exists $\tau \subset B$ enumerated into $\Phi_{\alpha_n}(i)$. If $i \in A$, then for any $\tau \subset B$ enumerated into $\Phi_{\alpha_n}(i)$, there is a demand issued $(\tau, i, \eta_0 \eta_1)$, such that $\eta_0 \subset \Psi_e(\tau)$ and η_1 is incomparable with $\Psi_e(\tau)$. Since $\Psi_e(B) = M$, mwe have $\eta_0 \subset M$, and there is a stage after which this demand is always implemented and not injured giving the required contradiction.

Finally, finally suppose that α_n is assigned requirement $(M)_e$. Out task is to show that, subsequent to the last initialisation of α_n, once T_{α_n} is non-empty, μ_s extends a leaf of T_α at every stage at which α_n is visited. Once we have shown this we are able to argue the minimality of M using Lemmas 3.1.3, 3.1.4 and 3.1.5, apply to M, and hence M is minimal.

Let s_0 be the first stage at which α_n is visited subsequent to its last initialisation. Let $s_1 > s_0$ be the stage at which we define $T_{\alpha_n}(\emptyset)$. Then at every subsequent stage $s \geq s_1$ at which α_n is visited $T_{\alpha_n}(\emptyset) \subseteq \mu_s$ and the implemented demands are precisely those which are implemented at stage s_1, together with possibly extra demands issued by nodes properly extending α_n on the construction tree, which are of the form $(\tau, 1, \eta_0, \eta_1)$ for η_0 and η_1 in T_{α_n}

$\square$

The end of Lemma 4.2.4 means that for all α either M lies on T_α which is either a computable Ψ_α splitting tree or for which above some initial segment of M contains no further Ψ_α splittings and so we are able to use Posner's trick (Lemma 3.1.5) which gives us that M is incomputable.

This completes the proof of Theorem 4.2.1

$\square$

Corollary 4.2.2 follows directly and so the conjecture of Cooper and Epstein from [CE87] is settled - incomputable $c.e.$ degrees satisfy the meet property.

Chapter 5

1-Generic Degrees and the Meet Property

First we show the existence of a 1-generic set using an oracle, and then give proofs for some of the basic (known) properties of 1-generics. Then, as for the minimal degrees in Chapter 3, we give a full approximation construction of a 1-generic which we will need for the main result of this chapter, Theorem 5.2.2 in Section 5.2. For more details on the introductory work in this chapter see [Ler83].

Theorem 5.0.5 *The exists a 1-generic set $A \leq \emptyset'$.*

Proof: We use an oracle for $\emptyset'$ to pick strings $\alpha_0 \subset \alpha_1 \subset \cdots \subset A = \bigcup_{i\in\omega} \alpha_i$ such that each α_i satisfies the following requirement

$$\mathcal{P}_i : \ \alpha_i \in W_i \vee (\forall \tau \supseteq \alpha_i)(\tau \notin W_i),$$

where W_i is the i^{th} c.e. set of finite strings.

Stage 0: If W_0 is empty set $\alpha_0 = \lambda$ otherwise set $\alpha_0 = \tau$ where τ is the least lexicographically least string in W_0.

Stage s+1: Use the oracle to check if $(\exists \tau \supset \alpha_s)[\tau \supseteq \text{ some } \tau' \in W_i]$. Consider the two possible answers:

Yes - Choose the least such τ and define $\alpha_{s+1} = \tau$. In this case α_{s+1} satisfies the first part requirement $\mathcal{P}_{s+1}$.

No - Define $\alpha_{s+1} = {\alpha_s}^{*}0$. In this case α_{s+1} satisfies the second.

Clearly there are no other possible options, and so $\mathcal{P}_i$ is satisfied for all i, hence A is 1-generic. We also only need access to a $\emptyset'$ oracle during the proof, and hence $A \leq_T \emptyset'$

□

Having shown the existence of a 1-generic set we now give some basic properties.

Theorem 5.0.6 *If A is 1-generic, then A is not computable.*

Proof: Suppose that B is an arbitrary recursive set, and let i be such that

$$\Psi_i(X;x) = 1 \Leftrightarrow \exists z B(z) \neq X(z),$$

and is undefined otherwise.

If $\Psi_i(A,x) \downarrow$, then clearly $A \neq B$. Suppose that $\Psi_i(A)(x)$ is undefined, then (by the genericity of A),

$$\exists \sigma \subseteq A (\forall \tau \supseteq \sigma)(\Psi_i(\tau;x) \uparrow).$$

This is a contradiction, as given an arbitrary σ, we are able to find $X \supset \sigma$ such that $\Psi_i(X;x) \downarrow$ by picking some n on which σ is undefined and letting X differ from B on it. So $\Psi_i(A;x) \downarrow$ and so $A \neq B$. Hence A is incomputable.

□

Theorem 5.0.7 *If A is 1-generic then it is GL_1, and hence is low if below $\emptyset'$.*

Proof: As A is 1-generic for every c.e. set of finite strings W there exists $\sigma \subset A$ such that either

1. $\sigma \in W$;
2. $\forall \tau \supset \sigma (\tau \notin W)$.

Consider the c.e. sets given by $V_i = \{\tau : \Psi_i(\tau; i)\downarrow\}$ for $i \geq 0$. Using an oracle for $A \oplus \emptyset'$ we are able to check for successive initial segments σ of A whether 1 and 2 holds (for σ and $W = V_i$ for any given i). Since, eventually either 1 or 2 must hold - and since $i \in A'$ if and only if former happens - it follows that $A \leq_T A \oplus \emptyset'$.

□

5.1 A 1-Generic by Full Approximation

In this section give a proof of the existence of a 1-generic degree, using a full approximation construction. We do this, as we will use a full approximation construction when we disucss 1-generics and the meet property in Section 5.2.

Although the notation is similar to that of the proof of Theorem 5.0.6 the proof is quite different. Where as the proof of Theorem 5.0.6 use a finite extension method, the following proof uses a full approximation method. Instead of asking an oracle for strings, we use a search process which is bounded at each stage and allow a module to injure modules of lower priority. This means that as the construction progresses our approximation at stage s to α_n associated with requirment $\mathcal{P}_n$ may change. If this happens we must discard all strings associated with modules of lower priority.

Theorem 5.1.1 *There exists a* 1*-generic degree.*

Proof [by full approximation]: We build a set A in stages $\{\alpha_s\}_s$, $lim_{s\to\infty}\alpha_s = A$. We have one type of requirement:

$$\mathcal{P}_i : \alpha_i \in W_i \vee (\forall\tau \supseteq \alpha_i)(\tau \notin W_i),$$

where W_i is the i^{th} c.e. set of finite substrings. Requirements are placed on a tree of strategies, and each module may be in two states - 0 and 1. The states correspond to the branches leaving a node on the tree. All modules start in state 0.

Naively, each module is given a finite string σ, a c.e. set W_i and acts as follows. While in state 0 it searches for an extension of σ which is in W_i. If such an extension is found then the module makes it an initial segment of A and moves into state 1. At each stage we bound the search that a module can do.

A module β working towards strategy $\mathcal{P}_i$, say, may only move from state 0 into state 1. When it does so, we access new modules on the tree of strategies for every strategy of lower priority than $\mathcal{P}_i$. When first accessed these modules are given associated strings. This is defined formally later.

The control path is the set of modules accessed during a stage.

The formal construction is as follows:

Each $\mathcal{P}_i$ module β (i.e., $|\beta| = i$) on the tree of strategies, has a string σ_β associated with it when it is first accessed, and β also has the set W_i associated with it. We let α_s be the approximation to A during stage s. At the end of a stage s we define α_s to be the longest σ_β associated with a module β on the control path during stage s. Define $A = lim_{s\to\infty}\alpha_s$. The true path is the right most path visited during the construction.

Stage 0: Set $\alpha_0 = \lambda$.

Stage $s+1$: The stage finishes and the next stage is begun when either a module changes state, or height $s+1$ of the tree of strategies is reached. Work through the modules on the control path acting as follows. When a module β with associated W_i is reached, and is in state 1, do nothing and pass control along the branch labeled 1. If β is in state 0, and has an associated string σ_β then search for a string (of length at most $s+1$) $\tau \supset \sigma_\beta$ which is in $W_i[s]$. If no such string is found, no action is performed and control is passed down branch 0. If such a string is found then σ_β is redefined to be an extension of τ longer than any string previously seen, the module is put into state 1 and the stage ends. If β does not have an associated string then associate $s_\beta = s_{beta^-} * 0$ with it, and end the stage.

The True Path is the left most Control Path visited infinitely often during the construction. Each module may only move from state 0 to state 1. If a module moves state it does so at some finite stage. It follows that the True Path is well defined.

To see that A is 1-generic. Assume otherwise, i.e., that for some module β, concerned with W_i is on the true path but not satisfied - so $\sigma_\beta \notin A$ but there exists some extension τ of σ_β such that $\tau \in W_i$. Let β be the module of highest priority for which this is the case. If this is the case, then at some finite stage we will find τ, and redefine σ_β to be such extension of τ - contradicting the assumption that β is not satisfied.

□

5.2 1-Generic Degrees and the Meet Property

The definition of 1-genericity can be extended as follows.

Definition 5.2.1 *A set A is n generic, for $n \geq 1$, if for every Σ_n set of strings S either*

1. $(\exists \sigma \subset A)[\sigma \in S]$*; or*

2. $(\exists\sigma \subset A)(\forall\tau \supseteq \sigma)[\tau \notin S]$.

In the late 1970s Jockusch [Joc77a] show that every degree strictly below a 2-generic degree satisfies the complemenetation property and in 1993 [Kum93] Kumabe furthered this by showing that for all $n \geq 2$, every n-generic has the complementation property. In [Joc77a] Jockusch also shows that for $n \geq 2$ no n-generic bounds a minimal degree, and together with Chong he showed that no 1-generic computable in $\mathbf{0}'$ (i.e., in the local degrees) bounds a minimal degree [CJ85] - meaning that we are unable to approach 1-generics and the meet property in the same way we approached c.e. degrees and the meet proprty.

Following Kumabe's paper a natural question to ask is *"Do 1-generics satisfy the complementation property?"*. In this section we go a step further and prove

Theorem 5.2.2 *There exists a 1-generic set D which does not satisfy the meet property.*

We construct a 1-generic set D (of degree $\mathbf{d}$) which does not have the meet property. To show that D does not have the meet property we construct a set $B <_T D$ (of degree $\mathbf{b}$) such that for all non-computable sets $C \leq_T D$ (of degree $\mathbf{c}$), $\mathbf{b} \wedge \mathbf{c}$ is non-computable.

5.2.1 Outline Proof

To achieve this we satisfy the following requirements for all i and j.

$\mathcal{P}_i : \exists\delta \subset D[\delta \in W_i \vee (\forall\tau \supset \delta)(\tau \notin W_i)]$.

$\mathcal{Q}_i : \Psi_i(B) \neq D$.

$\mathcal{R}_{\langle i,j\rangle} : \Psi_i(D)$ is non-computable $\rightarrow [\Gamma_i(\Psi_i(D)) = \Theta_i(B) \neq \Psi_j(\emptyset)]$.

Where for each i, Ψ_i is a given functional, and for all i we construct Γ_i and Θ_i.

Each $\mathcal{R}_{\langle i,j\rangle}$ works as a subrequirement of a larger requirement which is concerned with constructing Θ_i - how thi works in details is explained later. The $\mathcal{P}_i$ strategies ensure that D is 1-generic, the $\mathcal{Q}_i$ strategies ensure that $B \not\geq_T D$, and the latter combined with the $\mathcal{R}_{\langle i,j\rangle}$ strategies, show that D does not have the meet property (assuming $B \leq_T D$). It follows from the construction that $B \leq_T D$ but in order to make it easy to see we build a functional Λ such that $\Lambda(D) = B$.

As is standard we build functionals by adding axioms to them - these are triples consisting of an oracle string, an input and an an output. AS string is associated with an axiom it it extends the oracle string in the axiom. By τ is an *axiom free* extension of σ we mean that the only axioms associated with τ are also associated with σ.

As we proceed we will see that this consrtuction has several competing sets of requirements. In order to aid a conceptual understanding of the approach we are taking we give a fairly detailied naive construction. This makes the formal construction, which comes after, easier to follow.

We list our requirements in the following order of priority:

$$\mathcal{P}_0 > \mathcal{Q}_0 > \mathcal{R}_0 > \cdots > \mathcal{P}_n > \mathcal{Q}_n > \mathcal{R}_n > \cdots$$

During the construction we define B solely in terms of $\Lambda(D)$, so if $\delta \upharpoonright_n$ is fixed and $\Lambda(\delta \upharpoonright_n) \downarrow= \beta \upharpoonright_m$ then $\beta \upharpoonright_m$ is fixed.

Isolated Modules

First we outline the basic strategies. In this section we just consider how a module would act in isolation, if it did not have to consider the actions of other modules. Later we will discuss problems arising because of the interaction of the modules and then give a formal construction.

The module descriptions we give here will be adapted later for use on the tree of strategies. To simplify this process we include in the descriptions the points at which the modules will have to wait for other modules to act. This manifests itself as defining values and then checking conditions, as opposed to (more logically) checking conditions before assigning values. In the following we let δ be an initial segment of D, and β be an initial segment of B.

The $\mathcal{P}_i$ Strategies

$$\exists\delta \subset D[\delta \in W_i \vee (\forall\tau \supset \delta)(\tau \notin W_i)].$$

Each $\mathcal{P}_i$ module has a given string δ, say, and ais associated to W_i. It searches for an extension of δ which is in W_i. At each stage the search is bounded.

If such an extension of δ is found, then the module notes this, and makes it an initial segment of D. The module then performs no further actions. If such an extension is never found the module need not act.

The $\mathcal{Q}_i$ Strategies

$$\Psi_n(B) \neq D.$$

The $\mathcal{Q}_i$ strategies look to show that $\Psi_i(B) \neq D$. They do this by finding a witness to the inequality in the requirement.

Initially each module is given a pair of strings of equal length, δ_1 and δ_2, which extend δ the current approximation to D, which do not form a Λ splitting. Without loss of generality we may assume that $\Lambda(\delta_1) \supseteq \Lambda(\delta_2)$. An axiom free extension, $\Lambda(\delta) = \beta$ of

$\Lambda(\delta_1)$ is picked which is of the same length as δ_1 and axioms are added to Λ such that $\Lambda(\delta_1 * 0) = \Lambda(\delta_2 * 1) = \beta$. $\delta = \delta_1 * 0$ is then associated with Q_n, as is $m = |\delta| - 1$.

At subsequent stages s, where $\delta_s \supset \delta$, ((so $\Lambda(\delta_s) = \beta_s \supseteq \beta$) the module checks if $\Psi_i(\beta_s, m) \downarrow$. Once this has occurred, if $\delta(m) = 1 - \Psi_n(\beta_s, m)$ then successful diagonalisation has occurred and δ_s is now associated with this module.

If $\delta(m) = \Psi_i(\beta, m)$ then an axiom free extension of $\delta_2 * 0$ of length β_s, δ', is chosen. This extension is associated with the module, and axioms are added to Λ so that $\Lambda(\delta') = \beta_s$.

The $\mathcal{R}_{\langle i,j \rangle}$ Strategies

$$\Psi_i(D) \text{ is incomputable } \rightarrow [\Gamma_i(\Psi_i(D)) = \Theta_i(B) \neq \Psi_j(\emptyset)].$$

Initially the module has two associated strings δ and β, and an associated functional Ψ_i. First the module searches for two extensions δ' and δ'' of δ (of the same length) which form a Ψ_i splitting. At each stage the search is bounded, and if a splitting is never found then the module performs no other actions.

Once a splitting has been found the module picks a witness m and adds axioms to Γ_i such that $0 = \Gamma_i(\Psi_i(\delta'), m) \neq \Gamma_i(\Psi_i(\delta''), m) = 1$. The module picks two distinct extensions β' and β'' of β (of the same length as δ') and adds axioms into Θ_i so that $\Theta_i(\beta', m) = \Gamma_i(\Psi_i(\delta'), m) = 0$ and $\Theta_i(\beta'', m) = \Gamma_i(\Psi_i(\delta''), m) = 1$; and so that $\Lambda(\delta') = \beta'$ and $\Lambda(\delta'') = \beta''$. It sets δ' to be an initial segment of D, and so $\Lambda(\delta') = \beta'$ is now an initial segment of B.

At subsequent stages the module looks to see if $\Psi_j(\emptyset, m) \downarrow$, and if so whether $\Gamma_i(\Psi_i(\delta'), m) = \Psi_j(\emptyset, m) \downarrow$. If not the module need perform no further actions. If so the module sets δ'' as an initial segment of D, meaning $\Lambda(\delta'') = \beta''$ is an initial segment of B, and performs no further actions.

Module Interactions

The Γ_i and Θ_i functionals we construct are associated with module groups (i.e. for a given i, Γ_i is associated with $\mathcal{R}_{\langle i,j\rangle}$ modules for every value of j). The reason for this is that for a given Γ_i (or Θ_i) functional, if $\Psi_j(\emptyset)$ is total then (on some input) it must be unequal to $\Gamma_i(\Psi_i(D))$. If $\Psi_i(D)$ is incomputable then Γ_i must be total. We know that $\Gamma_i(\Psi_i(D))$ is not computable once the entire associated module group has been tested. We define the priority of a module group as the priority of its highest priority member - this module is called the *prime* module for (or in) the group. As the construction continues a module's approximation of D may change. This will, potentially, affect the value of $\Psi_i(D)$, for a given i, and hence the axioms we have enumerated of Γ_i.

We build Γ_i in response to what happens to $\Psi_i(D)$, as our approximations to D change, and we build Θ_i in response to Γ_i. During the construction we build various versions of the functionals Γ_i and Θ_i. We start to build a functional when we reach a module group's prime module. Each functional Ψ_i may be associated with several prime modules and several module groups. We require that the functionals Γ_i and Θ_i built by module groups associated with prime modules on the true path are total, and that for each i such a prime module exists. Movement of the control path on the tree of strategies below a prime module means that a new prime module will be accessed and new versions of the functionals Γ_i and Θ_i will start to be built. We label them Γ_i^α and Θ_i^α where α is the prime module for that group, on the current control path. When talking about a functional Γ_i we almost always drop the superscript, as by any mention of Γ_i we mean Γ_i^α for α, the prime module of highest priority on the current control path. We only build one version of the functional Λ.

Consider three nodes, α_1, α_2 and α_3, on the control path with the following order of priority $\alpha_1 > \alpha_2 > \alpha_3$ and with α_1 and α_3 R nodes in the same module group (i.e. $\mathcal{R}_{\langle i,j\rangle}$ modules which have the same i value but different j values) with prime module α and α_2 a Q_i module. The main concern we have is that the following will happen, meaning that (at least) one of the R module groups fails.

1. α_1 starts to enumerate Γ_i^α and Θ_i^α. It adds axioms such that $\Gamma_i^\alpha(\Psi_i(\delta), m_1) = 0$ and $\Theta_i^\alpha(\beta, m_1) = 0$.

2. α_3 (higher up the tree than α_1) finds a splitting for Ψ_i, δ' and δ'' above δ and enumerates axioms such that $\Gamma_i^\alpha(\Psi_i(\delta'), m_2) = 0$ and $\Theta_i^\alpha(\beta', m_2) = 0$ (where $\beta' \supset \beta$).

3. α_3 has to diagonalise, enumerating axioms such that $\Gamma_i^\alpha(\Psi_i(\delta''), m_2) = 1$ and $\Theta_i^\alpha(\beta'', m_2) = 1$, where $\beta'' \supset \beta$.

4. α_2 (between α_1 and α_3 on the tree of strategies) acts, forcing us to change our current approximation to D for δ'', to δ^*, but having to keep β'' the same. For the new δ^*, we have $\Psi_i(\delta^*) = \Psi_i(\delta')$ (up to input m_2) so $\Psi_i(\delta^*)$ is already associated with a Γ_i^α axiom. $\Gamma_i^\alpha(\Psi_i(\delta'), m_2) = 0$ - because of α_3. Similarly β'' is associated with a Θ_i^α axiom, so $\Theta_i^\alpha(\beta'', m_2) = 1$.

5. We are left in a situation where $\Gamma_i^\alpha(\Psi_i(D)) \neq \Theta_i^\alpha(B)$.

The crux of the problem in this example is how we pick a witness for α_2 (the $\mathcal{Q}_n$ requirement). We have to choose a witness so that we are able to move D as necessary without worrying whether or not modules of lower priority have already diagonalised (and so added in axioms which might cause $\Gamma_i(\Psi_i(D) \neq \Theta_i(B))$. To do this we look for two strings which form a splitting for every Ψ_i associated with a prime module of higher priority than the $\mathcal{Q}_n$ module we are working with. If when we move D we do so across such a splitting, then the new value of $\Psi_i(D)$, for any Ψ_i of higher prime priority, will not be associated with any Γ_i axioms above the point of the movement.

We are unable to look directly for pairs of strings which do not form a splitting for multiple different functionals. However if we are given pairs of strings (with certain properties

- outline below), each of which is not a splitting for a certain functional, we are able to combine them together to get a pair of strings which are not splitting for multiple functionals. Doing this allows us to build the splittings we need in order that whenever we move D we do so without causing $\Gamma_i(\Psi_i(D) \neq \Theta_i(B)$.

To build these splittings given a Ψ_i splitting $\tau_1, \tau_2 \supset \tau$ and two Ψ_j splittings $\tau_1', \tau_1'' \supset \tau_1$ and $\tau_2', \tau_2'' \supset \tau_2$ we pick a pair of strings (from amongst $\tau_1', \tau_1'' \supset \tau_1$ and $\tau_2', \tau_2'' \supset \tau_2$) which form both a Ψ_i splitting and a Ψ_j splitting. This is possible because both of τ_1' and τ_1'' will form a Ψ_i splitting with either of τ_2' and τ_2'' (as they extend opposite sides of a Ψ_i splitting), and both pairs independently Ψ_j split. Without loss of generality assume that $\Psi_j(\tau_1')$ is the longest for the given strings. At least one of τ_2' and τ_2'' must form Ψ_j splitting with τ_1', as although $\Psi_j(\tau_1')$ may extend (at most) one of $\Psi_j(\tau_2')$ and $\Psi_j(\tau_2'')$ if it does so it must be incompatible with the other. So if $\Psi_j(\tau_1') \supset \Psi_j(\tau_2')$ then τ_1' and τ_2'' are both Ψ_i and Ψ_j splitting. If $\Psi_j(\tau_1')$ extends neither $\Psi_j(\tau_2')$ nor $\Psi_j(\tau_2'')$, then τ_1' forms a splitting with both τ_2' and τ_2''.

Definition 5.2.3 *By* Combine Splittings*, we mean that, in a situation where τ_i', τ_i'' Ψ_i split and $\tau_{j,1}', \tau_{j,1}'' \supset \tau_i'$ Ψ_j split and $\tau_{j,2}', \tau_{j,2}'' \supset \tau_i''$ Ψ_j split, pick whichever pair of $\{\tau_{j,1}', \tau_{j,1}''\}$ and $\{\tau_{j,2}', \tau_{j,2}''\}$ form a splitting for both Ψ_i and Ψ_j to use as a new splitting.*

In the example above we were given the necessary splittings. The difficulty we have in using this process to create splittings for finitely many functionals is that we don't know whether splittings exist for a given Ψ above a given string, and if a splitting does exist we don't know when we are going to find it. So, we are unable to simply wait until we find a splitting before moving on.

It is also worth noting that we must also take into account that when we find a splitting, δ', δ'' for a given functional Ψ, previous modules may have enumerated axioms into Λ for either (or both) of δ', δ''. This is because we must keep enumerating Λ axioms, and this may happen before we know which Ψ splitting we will use, and may have no choice about the tails of the splitting.

Searching for Nested Splittings

To find the required nested splittings we introduce two new kinds of module, exactly how these fit into the tree of strategies will be discussed in the next section - for the moment we will consider them as isolated clusters of modules. The modules we introduce are S modules, which search for splittings, and C modules, which co-ordinate S modules.

S nodes and C nodes are linked together into clusters, containing one C node and at least one S node. We define the set of left descendants of a node α to be the set defined iteratively as follows: initially enumerate α into S, at subsequent stages of the iteration enumerate the left child of the greatest node in S into S and go to the next stage. If not such node exists finish the iteration. The left descendants of a C node are all S nodes. As the construction progresses, these clusters build approximations to the nested splittings required by Q modules.

Definition 5.2.4 *A node σ on the tree of strategies* believes *a splitting exists for Ψ if either all the ancestors of σ which have searched for a splitting for Ψ have found one, or none of σ's ancestors have searched for a Ψ splitting.*

Modules on the tree of strategies only enumerate axioms into Γ_i and Θ_i if they are on a part of the tree which believes that Ψ_i splittings exist. For each functional Γ_i we require lots of different splittings, and for each version of Γ_i (i.e., Γ_i^α for each prime module α) these splittings will be different. We enumerate the splittings into a set associated with each module α (not just prime modules), denoted $s(\alpha)$ - these are the strings which α is keeping track of. We will formally describe $s(\alpha)$ later.

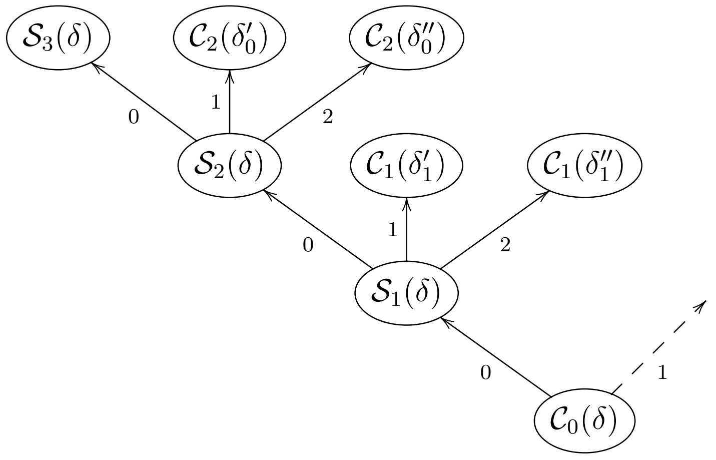
$\mathcal{S}_3(\delta)$
$\mathcal{C}_2(\delta_0')$
$\mathcal{C}_2(\delta_0'')$
0
1
2
$\mathcal{S}_2(\delta)$
$\mathcal{C}_1(\delta_1')$
$\mathcal{C}_1(\delta_1'')$
0
1
2
$\mathcal{S}_1(\delta)$
0
1
$\mathcal{C}_0(\delta)$

Figure 5.1: Example Cluster

Consider the example cluster given in Figure 5.1, which contains three S nodes. This cluster works above the string δ which the base node, α believes to be an initial segment of D. In this position on the tree of strategies we are building versions of Γ_0, Γ_1 and Γ_2. α does not believe that any other splittings (for functionals associated with modules of higher priority than α) will be found. The cluster hopes to find more splittings for Ψ_0,Ψ_1 and Ψ_2, and be able to nest those splittings. Remember that by our conventions Ψ_0 is the identity functional. We use the notation $s(\alpha)$ to mean the splittings that the node α is keeping track of - this is defined formally later.

When first passed control all the nodes in the cluster are in state 0 and control is passed along the left branches to S_3; at this point no other modules act. S_3 enumerates a Ψ_0 splitting (δ_0', δ_0''), passes this splitting to S_2 and moves S_2 into state 1.

The next time the cluster is passed control S_2 will pick one half of the splitting it has been given by S_3, δ_0' say, and searches for a Ψ_1 splitting above it. If S_2 fails to find such a splitting it passes control to C_2, the bottom of another C cluster, which does not believe that any further Ψ_1 splittings will be found. The new cluster will search for a new Ψ_0 splitting and then a Ψ_2 splitting extending this. If S_2 finds a splitting it enumerates it into $s(S_2)$, and enters state 2. The next time the cluster is passed control, S_2 searches for a

Ψ_1 splitting, above the other half of the splitting it was given by S_3, δ''_0. If S_2 fails to find the second splitting it passes control to the bottom of a different C cluster. Again this cluster does not believe that a Ψ_1 splitting exists above this point and searches for a new Ψ_0 splitting and then for a Ψ_2 splitting directly extending this. If S_2 does find a splitting above δ''_0, then it enumerates it into $s(S_2)$. It then combines this splitting with the splitting found when it was in state 1 to get a pair of strings which are splitting for both Ψ_0 and Ψ_1 - δ'_1, δ''_1. S_2 passes this new combined splitting down to S_1 and moves S_1 into state 1.

The next time the cluster is passed control S_1 will look for a Ψ_2 splitting above δ'_1. While it fails to find one S_1 passes control to a new C cluster. This cluster does not believe that Ψ_2 splits again and will just look for a pair of Ψ_1 splittings nested within a Ψ_0 splitting. Once S_1 has found a splitting it enumerates the splitting into $s(S_1)$ and enters state 2. In state 2 S_1 searches for a Ψ_2 splitting above δ''_1. As before if it fails to find a splitting S_2 passes control to the bottom of a new C cluster. If S_2 finds a splitting it enumerates it into $s(S_2)$. S_2 combines this splitting with the splitting it found while in state 1 to produce a splitting for Ψ_0, Ψ_1 and Ψ_2. S_2 passes the combined splitting down to C_0 and moves C_0 into state 1.

By convention, when we search for a splitting for Ψ above a string τ, we demand that the two halves of the splitting (if one is found) are of the same length, and longer than any string previously seen.

The reason that we use clusters like the ones described is that if a node is unable to find Ψ_i splitting for a particular i, then another cluster takes over - one which does not require a Ψ_i splitting for this i. This cluster then attempts to diagonalise D and B.

Let the example cluster look to diagonalise D and $\Psi(B)$. If splittings of the appropriate form are found for Ψ_0, Ψ_1 and Ψ_2, C_0 will pass them to a $\mathcal{Q}$ module with associated functional Ψ and diagonalisation is attempted. If no Ψ_1 splitting is found then one of the $\mathcal{C}_2$ modules will have control. These modules are at the base of a cluster which still wants to diagonalise D and $\Psi(B)$, but does not require a Ψ_1 splitting to do so.

The important point is that both $\mathcal{Q}$(or $\mathcal{R}$) modules attempt to provide a diagonalisation for D and B using the same functional Ψ, but require splittings to be found for different modules. If at any stage a new splitting is found by an $\mathcal{S}$ module then a different $\mathcal{Q}$ node, with the same associated functional, will be accessed.

Further Considerations

When searching for a Ψ_i splitting above a given string, we also require that the splitting found is not a Λ splitting, as otherwise we may find ourselves in a situation where different halves of the splitting already map to different β values - preventing diagonalisation.

If, above, an initial segment δ of D, every Ψ_i splitting (for a given i) is also a Λ splitting, then given an oracle for B we will be able to construct a Ψ_i nonsplitting tree containing D as a branch. Hence we will be able to compute $\Psi_i(D)$ from B, meaning that $\Psi_i(D)$ does not witness the failure of the meet property.

The formal details of how we do this are given in the next section, but we outline the process here. If the S module α which is looking for the splitting has associated functional Ψ_0, then it now looks for a Ψ_0 splitting which is not a Λ splitting. As before it enumerates the splitting into $s(\alpha)$, passes the splitting to its parent module and moves this module into state 1.

If the S module has associated functional Ψ_i (for any $i \neq 0$), and is in state 1 then it looks for a Ψ_i splitting extending one half of the splitting it has been given which is not a Λ splitting (δ_1 say) . If the module finds such a splitting δ_1', δ_1'' it enters state 2. When the module is first reached in state 2 it then searches for a Ψ_i splitting extending δ_2 which is not a Λ splitting. When we first enter state 2 and search for the splitting above δ_2, it is the first time that we have worked above δ_2, and so at this point no Λ axioms have been enumerated on extensions of it. If the module finds such a splitting δ_2', δ_2'' say, then as per Definition 5.2.3 it is able to pick a string from $\{\delta_1', \delta_1''\}$ and a string from $\{\delta_2', \delta_2''\}$ which form are splitting for both Ψ_i and the functional associated with α's daughter S module and which are also not Λ splitting. The process can then be repeated for every S module in the cluster until the base C module is given a nested splitting which is not Λ splitting.

Looking for splittings in this way does not affect our ability to diagonalise $\Psi_i(B)$ and D for all the i for which it is necessary, or to enumerate axioms into Λ so that $\Lambda(D) = B$, and so we still have $B <_T D$.

5.2.2 Tree of Strategies

The tree of strategies is made up of the following module types. All modules have branches leaving them labeled corresponding to their possible states, ordered numerically.

$\mathcal{P}$-nodes These nodes are concerned with making D 1-generic, and may be in two states 0 and 1. Both a $\mathcal{P}$ node's children are $\mathcal{C}$ nodes.

$\mathcal{Q}$-nodes These nodes are concerned with showing that $D \not\leq_T B$, and may be in two states 0 and 1. Both a $\mathcal{Q}$ node's children are $\mathcal{C}$ nodes.

$\mathcal{R}$-nodes These nodes are concerned with showing that if $\Psi(D) >_T \emptyset$ then if B and $\Psi(D)$ have a computable lower bound, then it is strictly above $\mathbf{0}$. They may be in 2 states, 0 and 1. All an R node's children are P nodes.

$\mathcal{S}$-nodes These nodes look for splittings and may be in three states 0, 1 and 2. An $\mathcal{S}$ node's left child is an $\mathcal{S}$ node, its other two children are $\mathcal{C}$ nodes.

$\mathcal{C}$-nodes These nodes organise $\mathcal{S}$ nodes and may be in two states 0 and 1. A $\mathcal{C}$ node's left child is an $\mathcal{S}$ node. A $\mathcal{C}$ nodes right child depends upon whether, when retracing the branch it is on, a $\mathcal{Q}$ node is reached before an $\mathcal{R}$ node or vice-versa. In the first case the $\mathcal{C}$-node's right child is an $\mathcal{R}$-node and in the second it is a $\mathcal{Q}$-node.

Each node α also has an associated set $I(\alpha)$, which contains the indices of all the functionals the node believes split.

Along every branch of the tree of strategies we order the $\mathcal{P}_i$, $\mathcal{Q}_i$ and $\mathcal{R}_{\langle i,j\rangle}$ nodes as follows.

$\mathcal{P}_i$-nodes These nodes are ordered along each branch by subscript.

$\mathcal{Q}_i$-nodes These nodes are ordered along each branch by subscript.

$\mathcal{R}_{\langle i,j\rangle}$-nodes These nodes are ordered along each branch by subscript with the extra proviso that $i \in I(\alpha)$, i.e. we don't list $\mathcal{R}_{\langle i,j\rangle}$ modules if a previous module on the branch has failed to find a Ψ_i splitting.

We make explicit the ordering of the $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ requirements as it makes it easier to follow the construction. As $\mathcal{C}$ and $\mathcal{S}$ nodes are responsible only for finding splittings we do not need to number them in this way. How they act depends only on their associated values. Each $\mathcal{C}$ node has infinitely many left $\mathcal{S}$ successors but only finitely many of them are ever accessed (why this is is covered in the formal construction).

The ordering of the $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ modules is as given earlier, but they are now interspersed with $\mathcal{C}$ and $\mathcal{S}$ nodes. The base node of the tree is $\mathcal{P}_0$, and this is the module of highest priority.

A module may only move from state 0 into state 1 or (if applicable) from state 1 into state 2

Enumeration of Functionals

We break the enumeration of axioms down into two cases. As our approximation to D changes we have complete control over the Λ axioms we enumerate. Similarly we have complete control over Θ_i axioms. We do not want to be in a position where every proper extension of a string σ is associated with more axioms than σ itself, as this could cause problems with our functionals becoming unequal. The key to ensuring this does not happen is to ensure that for no string σ are both σ^*0 and σ^*1 used as oracles for any enumerated axioms. To do this every time we search for a string we require it to be strictly longer than any previously seen - so between stages there is no problem. Within a stage if we are enumerating more than one axiom we ensure that if $|\sigma^*0| = |\sigma'^*1|$ and we wish to enumerate axioms for both then $\sigma \neq \sigma'$. This means that at all subsequent stages when we go to enumerate axioms we will be able to do so, without worrying that the functionals are already defined on all extensions of a string.

As we alter D we have no control over $\Psi_i(D)$, and so are unable to do the same for Γ_i. To get around this we ensure that for all δ, if $\Lambda(\delta) = \beta$ and $\Psi_i(\delta) = \psi$, then $\Gamma_i(\psi) \supseteq \Theta_i(\beta)$.

$\mathcal{Q}_i$ nodes may enumerate axioms into Λ and Γ_i, and R_i nodes may enumerate axioms into Γ_i, Θ_i and Λ. $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{S}$ modules do not enumerate axioms into any functional.

At stage s we only enumerate axioms into Λ which have oracle strings of length greater than s. If the Ψ_i splitting we have found is of length less than s we arbitrarily extend it to be of the appropriate length. In this way when enumerating the Ψ_i nonsplitting tree, if necessary, at stage s we are able to look at strings shorter than s as candidates for enumeration knowing that at later stages no Λ axioms will be enumerated on them, causing a splitting to occur.

A $\mathcal{Q}$ module α is passed a splitting by its parent $\mathcal{C}$ node, α^-, and enumerates axioms into Λ mapping both halves of the splitting to the initial segment of B associated with α^-. At later stages if α is able to diagonalise, using a longer initial segment of B, then it adds axioms into all the necessary Γ_i (associated with prime modules of higher priority) so that $\Gamma_i(\Psi_i(\delta)) = \Theta_i(\beta)$.

An $\mathcal{R}$ module α is also passed a splitting by its parent $\mathcal{C}$ node. The $\mathcal{R}$ module picks an axiom free extension of both strings in the splitting and two distinct axiom free extensions of the initial segment of B, associated with α^- and adds axioms into Λ so that one half of the extended splitting maps to each. The $\mathcal{R}$ module extends Γ_i by one (but so that the output is different when different halves of the splitting are used as an oracle), and adds axioms into Θ_i to match.

5.2.3 Formal Construction

We break this into two sections, one containing the construction and one detailing how, we use an oracle for B to build a Ψ_i nonsplitting tree if we are unable to find a Ψ_i splitting which is not Λ splitting.

Every module starts in state 0. A module passes control down the tree, along the branch according to the state it is in. We describe modules handed control during a stage as being on the control path, only these modules are able to act during a stage. The true path is the rightmost control path visited during the construction.

Each module α has a string δ_α associated with it, this is the module's current approximation to D. If α is passed control at stage $s+1$ for the first time, then initially we let $\delta_\alpha = \delta_s$, where δ_s is the initial segment of D built by the end of stage s. At later stages it is possible that α will change the value of δ_α. Modules also have a string β_α associated with them. This is defined to be $\Lambda(\delta_\alpha)$, and is the module's current approximation to B.

At the end of a stage s+1 we define $\delta_{s+1} = \delta_\alpha$ for the longest δ_α, associated with a module α on the control path during this stage. We define $\beta_{s+1} = \Lambda(\delta_{s+1})$. Along a branch of the tree, for two modules if $\alpha \subseteq \alpha'$, we may only have $\delta_\alpha \subseteq \delta_{\alpha'}$ and $\beta_\alpha \subseteq \beta_{\alpha'}$.

A stage always ends when a module changes state. A stage may also end if instructed to do so by a module, this occurs after a module has had its associated values changed.

Every module α has an associated set $I(\alpha)$ of the indices of the Ψ_i functionals it believes split. The sets of indices are used in addition to the sets of functionals described later to monitor which splittings a module is looking for. If a module α which is looking for a Ψ_i splitting fails to find one, then it removes i from $I(\alpha)$. No module above α on the tree accessed while $i \notin I(\alpha)$ will look for a Ψ_i splitting. When next reached the module will continue to look for a Ψ_i splitting, and if one is found i will be re-enumerated into $I(\alpha)$. If i is re-enumerated into $I(\alpha)$ then a new branch of the tree will be accessed and some of these modules will look for a Ψ_i splitting. When it is first reached a module sets $I(\alpha) = I(\alpha^-)$ - as if its parent has failed to find a splitting the appropriate index will already have been removed from $I(\alpha^-)$.

Each $\mathcal{Q}$, $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{C}$ module α has a pair of strings associated with it which form a splitting for one or more Ψ_i. This pairs is denoted $s(\alpha)$. $\mathcal{Q}$ and $\mathcal{R}$ modules will be given this pair by their parent $\mathcal{C}$ module. A $\mathcal{C}$ module will be given $s(\alpha)$ by its daughter $\mathcal{S}$ module. For ease of notation when discussing an $\mathcal{R}_{\langle i,j\rangle}$ node with associated strings $s(\alpha) = \{\delta_1, \delta_2\}$, we will refer to the ($\Theta_i$ splitting) strings $\Lambda(\delta_1)$ and $\Lambda(\delta_2)$ as $s_\beta(\alpha) = \{\beta_1, \beta_2\}$. An $\mathcal{S}$ module has two extra pairs of strings associated with it - these are denoted $s_i(\alpha)$ for $i \in \{1, 2\}$, and are used to build the nested splittings.

$\mathcal{C}$ nodes have a set $F(\alpha) = \{\Psi_0, \cdots, \Psi_n\}$ of functionals that the cluster they are the base of will look to find nested splittings for. Each $\mathcal{S}$ node has a set $F(\alpha)$ of functionals, containing the functionals that they (and their left successors) still need to find splittings for. These sets are ordered by subscript from least to greatest. For an $\mathcal{S}$ whose parent is a $\mathcal{C}$ node $F(\alpha) = F(\alpha^-)$, for all other $\mathcal{S}$ nodes $F(\alpha) = F(\alpha^-) \setminus \{\Psi_i\}$ for the greatest i in $F(\alpha^-)$. When $F(\alpha) = \{\Psi_0\}$, we call the $\mathcal{S}$ node a *special $\mathcal{S}$ node*. This $\mathcal{S}$ node is guaranteed to find a splitting for all the functionals in $F(\alpha)$ (as Ψ_0 is the identity functional), and so no nodes further along the branch need be accessed. For a $\mathcal{C}$ node $F(\alpha) = \{\Psi_i : i \in I(\alpha^-) \text{ and } i \leq j \text{ where } j \text{ is the largest index of a prime module on the control path below } \alpha\}$.

$\mathcal{P}_i$ nodes have associated with them computably enumerable sets of strings. For a $\mathcal{P}$ node with index i this is the i^{th} c.e. set W_i.

$\mathcal{Q}$ and $\mathcal{R}$ nodes have an associated diagonalisation value, denoted $n(\alpha)$. This is initially undefined.

At stage 0 we have $\delta_0 = \beta_0 = \lambda$, $\Lambda = \emptyset$ and $\Gamma_i = \Theta_i = \emptyset$ (for all i) and all values and sets associated with modules are initially 0, with the exception of $I(\alpha) = \omega$ - where α is the base module of the tree. We describe how each module α acts if given control at stage $s + 1$ - we will refer to these instructions in the construciion whch follows.

$\mathcal{P}_i$ Strategies

State 0

1. Search for a string τ (of length at most $s + 1$) extending δ_α in W_i. If no such string is found perform no further actions.
2. If such a string is found then define δ_α to be an extension of τ, longer than any seen before which is Λ axiom free (above τ).
3. Enter State 1.

State 1

1. Do nothing.

$\mathcal{Q}_i$ Strategies

State 0

1. If this is the first time visiting this module, act as follows, otherwise go to instruction 2. Set $s(\alpha) = s(\alpha^-)(= \{\delta_1, \delta_2\})$. Assume without loss of generality that $\Lambda(\delta_1) \supset \Lambda(\delta_2)$ and pick an axiom free extension, β_α, of $\Lambda(\delta_1)$, of the same length as δ_1. Add axioms to Λ such that $\Lambda(\delta_1 * 1) = \Lambda(\delta_2 * 0) = \beta_\alpha$. Define $\delta_\alpha = \delta_1 * 1$ and set $n(\alpha) = |\delta_\alpha| - 1$.

   Finish the stage.

2. Check if $\Psi_i(\beta_s, n(\alpha)) \downarrow$. If so we consider two possibilities

   (a) $\delta_\alpha(n(\alpha)) \neq \Psi_i(\beta_s, n(\alpha))$. In this case we do not need to alter anything. Set $\delta_\alpha = \delta_s$ and enter state 1.

   (b) $\delta_\alpha(n(\alpha)) = \Psi_i(\beta_s, n(\alpha))$. In this case we redefine δ_α to be an axiom free extension of $\delta_2 * 0$ of length at least $|\beta_s|$, and set $\delta_{s+1} = \delta_\alpha$. Add axioms so that $\Lambda(\delta_\alpha) = \beta_\alpha$, add axioms to Γ_i as necessary for all i associated with prime modules of higher priority so that $\Gamma_i(\Psi_i(\delta_2)) = \Theta_i(\beta_s)$ and enter state 1.

   If $\Psi_i(\beta_s, n(\alpha)) \uparrow$ do nothing.

<u>State 1</u>

1. Do nothing.

$\mathcal{R}_{\langle i,j \rangle}$ Strategies

<u>State 0</u>

1. If this is the first time we are visiting this module act as follows, otherwise go to instruction 2.

(a) Define the $\delta_1, \delta_2 \in s(\alpha)$ to be Λ axiom avoiding extensions of δ' and $\delta'' \in s(\alpha^-)$, such that δ_1 and δ_2 are of the same length, and longer than any previously seen.

(b) Let β be the longest string which is an initial segment of both $\Lambda(\delta_1)$ and $\Lambda(\delta_2)$ and pick two distinct extensions of β, β_1 and β_2, which are of the same length as δ_1.

(c) Find the least m for which $\Gamma_i(\Psi_i(\delta_\alpha), m) \uparrow$ and set $n(\alpha) = m$.

(d) Set $\Gamma_i(\Psi_i(\delta_1), n(\alpha)) = 0$.

(e) For functionals associated with prime modules of higher priority add axioms as necessary, (i.e., for all functionals Γ_k, Θ_k, Λ_k which, if axioms were not re-enumerated, would be shorter than at the end of the previous stage i.e., axiom are added to Θ_k and Γ_k so that for all $m \leq n(\alpha)$, $\Theta_k(\beta_1, m) = \Gamma_i(\Psi_k(\delta_1), m) = 0$ and $\Theta_k(\beta_2, m) = \Gamma_i(\Psi_k(\delta_2), m) = 1$). Add axioms to Λ so that $\Lambda(\delta_1) = \beta_1$ and $\Lambda(\delta_2) = \beta_2$.

(f) Redefine $\delta_\alpha = \delta_1$

2. Check if $\Psi_j(\emptyset, n(\alpha)) \uparrow$. If so, then we perform no further actions, and move to the next module. Otherwise we continue.

3. If $\Psi_j(\emptyset, n(\alpha)) \downarrow= 1$, then the module enters state one (and we finish the stage).

4. If $\Psi_j(\emptyset, n(\alpha)) \downarrow= 0$, then set δ_α to be an axiom free extension of δ_2 and pick an axiom free extension of β_2 and add axioms into Λ so that $\Lambda(\delta_\alpha)$ equals this extension. Add axioms to Γ_j as necessary for all j associated with prime modules of higher priority so that $\Gamma_i(\Psi_j(\delta_2)) = \Theta_j(\beta)$ and enter state 1.

State 1

1. Do nothing.

Special $\mathcal{S}$ Modules

(These are $\mathcal{S}$ modules whose only associated functional is the identity functional.)

1. Search for a Ψ_0 splitting δ_1, δ_2 extending δ_α, which is not a Λ splitting. Set $s(\alpha^-) = \{\delta_1, \delta_2\}$ and move α^- into state 1.

Normal $\mathcal{S}$ Modules

State 0

1. Pass control along branch 0.

State 1

1. If this is the first time we are visiting this module act as follows, otherwise go to instruction 2. Set $\delta_\alpha = \delta_1 \in s(\alpha)$.
2. Search for a Ψ_i splitting extending δ_α for Ψ_i in $F(\alpha)$ with greatest subscript, which is not a Λ splitting. If no such splitting is found set $I(\alpha) = I(\alpha) \setminus \{i\}$, and pass control down branch 1. If such a splitting δ_1', δ_2' is found enumerate this into $s_1(\alpha)$ enumerate i into $I(\alpha)$ (if necessary), redefine δ_α to be an extension of $\delta_2 \in s(\alpha)$ which is not a Λ splitting with the longer of $\Lambda(\delta_1')$ and $\Lambda(\delta_2')$ and enter state 2.

State 2

1. Search for a Ψ_i splitting extending δ_α for the greatest Ψ_i in $F(\alpha)$, which is not a Λ splitting. If no such splitting is found set $I(\alpha) = I(\alpha) \setminus \{i\}$, and pass control down branch 2. If such a splitting δ_1'', δ_2'' is found enumerate this into $s_2(\alpha)$ enumerate i into $I(\alpha)$, redefine $\delta_\alpha = \delta_2 \in s_1(\alpha)$ and go to instruction 2.

2. *Combine*[1] $s_1(\alpha)$ and $s_2(\alpha)$ into a new splitting, and set $s(\alpha^-)$. to be this splitting. Move α^- into state 1.

$\mathcal{C}$ modules

State 0

1. Pass control down branch 0.

State 1

1. Pass control down branch 1.

Construction

We finish a stage whenever a module changes state. We also finish a stage if instructed to by a module. If no module changes state during a stage or has associated values redefined, the stage finishes when all of the actions described below have been completed. Once the construction has completed one stage it moves on to the next one. We bound the height on the tree of strategies we may access by the stage number.

Stage 0**:** Set $\delta_0 = \lambda$, $\beta_0 = \lambda$, $\Lambda = \emptyset$ and $\Gamma_i = \Theta_i = S_i = \emptyset$ for all i. Set all values and sets associated with modules to be zero or empty, with the exception of $I(\alpha) = \omega$ - where α is the base module of the tree.

Stage $s{+}1$**:** Work through the modules on the control path in order (from highest to lowest priority), acting on their instructions. If a module is able to act it does so as described, if a module is unable to act it passes control down the appropriate branch to the next module.

If no module is able to act, then the stage finishes and we move on to the next stage.

[1]c.f. Definition 5.2.1

Enumeration of Nonsplitting Trees

Here we describe how to enumerate T_i, a Ψ_i nonsplitting tree, if above an initial segment of D every Ψ_i splitting is also a Λ splitting, using an oracle for B.

If we need to build a Ψ_i nonsplitting tree it is because after some stage in the construction and above some intial segment of D no further Ψ_i splittings, which are Λ nonsplitting are found. We assume without loss of generality that we are at this stage. We may do this as the nonsplitting trees we build have no effect on the construction - they are not used by the construction in any way. The building of these trees does, however, follow the stages in the construction.

Assume that after stage s of the construction no further splittings of the appropriate form are found for Ψ_i, and at this stage the approximation to D is δ_s. At stage s enumerate λ and δ_s into T_i the Ψ_i nonsplitting tree.

At all subsequent stages s', search for the least extension τ of δ_s not currently in T_i, of length at most $|s'|$, such that $\Lambda(\tau)$ is an initial segment of B - use our oracle for B for this. If we find such a τ we enumerate it into T_i, and wait for the next stage to finish. If we do not find such a τ we enumerate nothing and wait for the next stage to finish. All searches are bound by the stage number, as is the length of any string enumerated into the tree.

We perform this process for all Ψ_i such that no further Ψ_i splittings which are Λ nonsplitting will be found.

Note that this does not affect our ability to diagonalise D against $\Psi_j(B)$, as some other Q module (which requires fewer Ψ splittings) will perform the diagonalisation.

5.2.4 Verification

In the verification we have to show that the construction is well defined and that each stage terminates, we also have to argue that the sets and functionals built have the required properties.

That a true path through the tree of strategies exists is clear - as each module is only able to change state at most twice, and cannot return to a state it has been in previously. If a module acts, it does so at stage s for some finite s - otherwise it stays in the same state forever. This means that every module that acts is at some stage the module of highest priority which still has to act. For a given module we do not know when this stage is, but the fact that it exists is important.

It is also clear that all of the instructions given in the construction are feasible. At no point do we perform an unbounded search, or require the use of an oracle we do not have access to. As we are using the convention that if at stage s a computation requires more than s steps then we assume (for the moment) that the computation is undefined and the process halts after s steps, all searches and computations terminate and every stage finishes in some finite time.

Claim 5.2.5 *For all i at every stage $s \geq 0$, for all δ_s if $\Lambda_s(\delta_s) = \beta_s$ and $\Psi_{i,s}(\delta_s) = \psi_s$, then $\Gamma_{i,s}(\psi_s) \supseteq \Theta_{i,s}(\beta)$.*

Proof: We prove this by induction on the stage number.

Stage 0: $\delta_0 = \beta_0 = \lambda$, $\Lambda = \emptyset$ and $\Gamma_{i,0} = \Theta_{i,0} = \emptyset$ for all i and so the result holds.

Stage s+1: Assume the result holds at the end of all stages $\leq s$ and prove that if this is the case then it also holds at the end of stage $s + 1$.

If no module acts, then the induction is trivial.

If a P module acts it moves δ_s above $\delta_\alpha \subseteq \delta_s$ to δ_{s+1}, which is longer than any approximation to D previously seen. There is a maximal initial segment of δ_{s+1}, δ, which is part of a Λ axiom, by the induction hypothesis $\Lambda(\delta) = \beta$ and $\Gamma_{i,s}(\Psi_{i,s}(\delta)) \supseteq \Theta_{i,s}(\beta)$ for all i. P modules do not add any axioms to any Γ nor Θ; or to Λ, and so at the end of the stage $\Gamma_{i,s+1} = \Gamma_{i,s}$; $\Theta_{i,s+1} = \Theta_{i,s}$, $\Lambda_{s+1} = \Lambda_s$ and for all i, for all $\delta \subset \delta_{s+1}$ such that $\Lambda_{s+1}(\delta) = \beta$ (an initial segment of β_{s+1}), $\Gamma_{i,s+1}(\Psi_{i,s+1}(\delta)) \supseteq \Theta_{i,s+1}(\beta)$ and so the induction holds.

Note that for some i there may be strings $\delta^{\ddagger}$ such that $\delta \subset \delta^{\ddagger} \subseteq \delta_{s+1}$ for which $\Psi_{i,s+1}(\delta^{\ddagger})$ is part of a Γ_i axiom but $\Lambda(\delta^{\ddagger})$ is not defined (yet) and so this does not matter. Note also that if $\delta^{\ddagger} \supset \delta$ then $\Gamma_{i,s+1}(\Psi_i(\delta^{\ddagger})) \supset \Gamma_{i,s+1}(\Psi_{i,s+1}(\delta))$ if they are both defined.

For reasons similar to those for $\mathcal{P}$ modules, if a $\mathcal{S}$ module enumerates a splitting, and extends δ_s, no axioms are enumerated and the induction holds. The details of this are omitted.

We consider all the possibilities for the actions of a $\mathcal{Q}$ module separately. We start with the module being in state 0 and instruction 1 acting. We may assume without loss of generality that $\Lambda_s(\delta_1) \supseteq \Lambda_s(\delta_2) \supseteq \Lambda_s(\delta_s)$. We show that if $\Lambda_{s+1}(\delta_2)$ violates the inductive condition, then so does $\Lambda_s(\delta_2)$ - contradicting the induction hypothesis. Suppose the $\mathcal{Q}$ module acts in state 1 to extend $\Lambda_s(\delta_2)$ to be equal to $\Lambda_s(\delta_1)$ and for contradiction let i be the least such that the extension of $\Lambda_s(\delta_2)$ violates the inductive condition (via $\Gamma_{i,s}$ or $\Theta_{i,s}$). Note that if $\Lambda_s(\delta_2) = \Lambda_s(\delta_1)$ then the induction trivially holds.

We are in a position where Γ_i and Θ_i axioms have been enumerated by another cluster on the tree of strategies.

δ_1 and δ_2 are Ψ_i splitting, so $\Psi_i(\delta_1) \neq \Psi_i(\delta_2)$. The concern we have is that $\Theta_{i,s}(\Lambda_s(\delta_1)) = \Gamma_{i,s}(\Psi_{i,s}(\delta_1)) \neq \Gamma_{i,s}(\Psi_{i,s}(\delta_2))$ - as $\Psi_{i,s}(\delta_2) = \Psi_{i,s}(\delta)$ for some δ associated with some previously accessed cluster - and that by extending $\Lambda_s(\delta_2)$ we will violate the induction condition. By considering the points at which axioms for Γ_i and Θ_i could have been enumerated we see that if the inductive condition is violated by such an extension, then in fact it has been violated at some previous stage.

We are in a situation where δ is incompatible with δ_2 (if $\delta \subset \delta_2$ it is easy to see that the situation we are concerned with does not arise).

If $\Psi_{i,s+1}(\delta_2)$ is comparable with $\Psi_{i,s+1}(\delta)$ for some previously accessed δ, for which $\Gamma_{i,s}(\Psi_{i,s+1}(\delta))$ is already defined then δ is part of some Ψ_i splitting which was previously found. Let δ' be the other half of this splitting.

Let τ be the longest initial segment of δ, δ', δ_1 and δ_2 and let τ' be the longest initial segment of δ and δ' and τ'' be the longest initial segment of δ_1 and δ_2. We have that $\Psi_{i,s+1}(\tau')$ and $\Psi_{i,s+1}(\tau'')$ are comparable and $\Psi_{i,s+1}(\delta')$ and $\Psi_{i,s+1}(\delta_1)$ are incomparable (as if the both $\Psi_{i,s+1}(\delta')$ and $\Psi_{i,s+1}(\delta_1)$ are comparable and $\Psi_{i,s+1}(\delta)$ and $\Psi_{i,s+1}(\delta_2)$ are comparable then the induction follows).

Consider when axioms are enumerated into Γ_i and Θ_i. If $\Theta_{i,s}$ has axioms enumerated into it for some extension of $\Lambda_s(\tau'')$ then the induction follows as we are in a position where $\Psi_{i,s}(\tau'')$ must be part of a $\Gamma_{i,s}$ axiom, and so if we are are unable to extend $\Lambda_s(\delta_2)$ as desired it must be because at some prior stage the inductive condition was violated. If $\Theta_{i,s}$ does not have axioms enumerated into it for some extension of $\Lambda_s(\tau'')$ then we can extend $\Lambda_s(\delta_2)$ as required as the extension of β_α picked by the $\mathcal{Q}$ is axiom free for $\Theta_{i,s}$ and so the induction holds.

If a $\mathcal{Q}$ module acts in state zero instruction two (a), then nothing is altered and the induction holds. If instruction (b) acts then we are just re-enumerating Θ axioms (which at the end of stage s did not violate the induction hypothesis) onto a string which is axiom free above $\Lambda_s(\delta_s)$, and so the induction holds.

If an $\mathcal{R}$ module acts on instruction 1 in state 0, then for reasons similar to those for $\mathcal{Q}$ modules the induction holds, the details are omitted. Instructions 2 and 3 of state 0 make no alterations to any axioms or strings and so the induction holds here.

If instruction 4 acts then we argue that the induction holds using similar reasons to those used for showing it holds when Λ_s is extended, in instruction one of a $\mathcal{Q}$ module.

$\square$

Claim 5.2.6 *If we are unable to diagonalise D and $\Psi_i(B)$, then $\Psi_i(D) \leq_T B$*

Proof: Assume that after stage s of the construction no further Ψ_i splittings which are not Λ splittings are found and so we are unable to diagonalise. If this is the case we look to build a Ψ_i non-splitting tree computably in B, which contains D as a branch.

Suppose that at stage s the approximation to D is δ_s. At stage s enumerate λ and δ_s into T_i the Ψ_i nonsplitting tree.

At all subsequent stages s', search for the least extension τ of δ_s not currently in T_i, of length at most $|s'|$, such that $\Lambda(\tau)$ is an initial segment of B - use our oracle for B for this. If we find such a τ we enumerate it into T_i, and wait for the next stage to finish. If we do not find such a τ we enumerate nothing and wait for the next stage to finish. All searches are bound by the stage number, as is the length of any string enumerated into the tree and hence we computably build a Ψ_i non splitting tree with D as a branch, and so by a similar argument to Lemma 3.1.4 $\Psi_i(D) \leq_T B$.

$\square$

Claim 5.2.7 *B is total and for all j, if $\Psi_i(D)$ is total, then both $\Gamma_i(\Psi_i(D))$ and $\Theta_i(D)$ are total.*

Proof: When first given control $\mathcal{R}$ a module α on the control path is given an associated $\beta_\alpha(= \Lambda(\delta_\alpha))$, of some length n. α may not move β_α below length n. $\mathcal{Q}$ and $\mathcal{R}$ modules extend our approximation to β_α, and we access infinitely many of them on the true path, and so we can see that $B = lim_{\alpha \to \omega} \beta_\alpha$ is infinite.

If $\Psi_i(D)$ is total, then eventually for all $j \in \omega$ the module $\mathcal{R}_{\langle i,j \rangle}$ is the module of on the true path of highest priority yet to act. At this point both $\Gamma_i(D)$ and $\Theta_i(D)$ are extended up to a certain height, n (strictly greater than previously seen on the true path) and are never again injured below this height.

Claim 5.2.8 *For all j, if Ψ_j is total, then if $\Psi_i(D)$ total for some input n, $\Gamma_i(\Psi_i(D,n)) \neq \Psi_j(n)$.*

Proof: For a given Γ_i, if this is not true then there is some minimum j for which Ψ_j witnesses the failure. We assume that this i and j are being dealt with by the module R on the True Path.

As noted in the introduction to this section, eventually $\mathcal{R}$ is the module of highest priority still to act. After it has been injured for the last time R finds a splitting and chooses a witness to try and show that $\Gamma_i(\Psi_i(D)) \neq \Psi_j$, say argument n.

If $\Psi_j \downarrow$ at this stage then we can pick diagonalise $\Gamma_i(\Psi_i(D))$ and Ψ_j straight away, giving a contradiction in our assumption

If $\Psi_j(n) \uparrow$ at this stage then as we need to keep extending $\Gamma_i(\Psi_i(D))$ we choose an output for $Gamma_i(\Psi_i(D), n)$. Eventually (as it is total) $\Psi_j(n) \downarrow$. At this stage we are able to diagonalise.

$\mathcal{R}$ was the module of highest priority still to act, and so is never again injured, and n witness that $\Gamma_i(\Psi_i) \neq \Psi_j$ - contradicting that j was the least such failure.

□

Claim 5.2.9 *$B <_T D$.*

Proof: The $\mathcal{Q}_i$ strategies are concerned with showing that $\Psi_i(B) \neq D$, and hence $B \not\geq_T D$. Let i be the least such that $\Psi_i(B) = D$ (by assumption here, Ψ_i is total). Let $\mathcal{Q}$ be the module on the true path which is dealing with this i. At some stage of the construction $\mathcal{Q}$ was the module of highest priority yet to act. After it was been injured for the last time $\mathcal{Q}$ chose a witness n to try and show that $\Psi_i(B,n) \neq D(n)$. At some stage (later) $D(n) \downarrow$ (as D is defined on every input) and at some stage (possibly later) $\Psi_i(B,n) \downarrow$ (as it is total). At this stage we were able to move D, while leaving B the same and hence force $\Psi_i(B) \neq D$. Contradicting the assumption that i was the least failure.

This shows that $D \leq_T B$. As we are able to compute B from D via Λ, we know that B and D are not incompatible, and so B is strictly less than D.

$\square$

Claim 5.2.10 *D is 1-generic*

*Proof:*Assume otherwise, i.e., that module α, concerned with W_i, is on the true path but not satisfied - so $\delta_\alpha \notin W_i$ but there exists some extension τ of σ_α such that $\tau \in W_i$. Let α' be the module of highest priority for which this is the case. If this is the case, then at some finite stage we will find τ, and redefine σ_α to be such extension of τ. If α is on the true path, then (as modules may only change state once), not module of higher priority on the True Path beneath α acts at any subsequent stage. This means that at no stage after s' is α injured. Contradicting the assumption that α is not satisfied.

$\square$

These claims show that our construction is indeed of a 1-generic set D, which does not have the meet property, and so conclude the proof of Theorem 5.2.2

$\square$

As previously noted in Chapter 4, for the final proof in this thesis we were unable to proceed by building a minimal degree. Chong and Downey showed that there exist minimal degrees that are bound by 1-generics between $\mathbf{0}'$ and $\mathbf{0}''$ [CD90], and also that there exist minimal degrees below $\mathbf{0}'$ which are bound by no 1- generic [CD89]. Indeed in the later paper they state:

> ..the interplay between 1-genericcs and minimal degrees appears to be fairly complex.

and

> ...there is more work to be done in this area.

Another interesting open problem, asked by Slaman and Steel [RT99], is whether $\mathbf{0}'$ is the least degree such that all degrees above satisfy the complementation property.

Bibliography

[AD77] A. A. Al-Daffa'. *The Muslim Contribution to Mathematics.* Croom Helm/Humanities Press, 1977.

[ASF06] K. Ambos-Spies and P.A. Fejer. Degrees of Unsolvability. 2006.

[CD89] C.T. Chong and R. Downey. Defining the Turing Jump. *Math. Proc. Camb. Phil. Soc.*, 105(5):211 – 222, 1989.

[CD90] C.T. Chong and R. Downey. Defining the Turing Jump. *Ann. Pure and Appl. Logic*, 48:215 – 225, 1990.

[CE87] S. B. Cooper and R. L. Epstein. Complementing Below Recursively Enumerable Degrees. *Ann. of Pure and Appl. Logic*, 34:15 – 32, 1987.

[CJ85] C.T. Chong and C.G. Jockusch. Minimal degrees and 1-genrics below $\mathbf{0}'$. In M.M. Richter, E. Börger, W. Oberschelp, B. Schinzel, and W. Thomas, editors, *Proceedings of Aachen Logic Conference*, Berlin, 1985.

[Coh63] P.J. Cohen. Set Theory and the Continuum Hypothesis. *Proc. Natl. Acad. Sci. USA*, 55(6):1143 – 1148, 1963.

[Coo72] S. B. Cooper. Degrees of Unsolvability Complementary between Recursively Enumerable Degrees. *Ann. of Math. Logic*, 4(1):31 – 73, 1972.

[Coo74] S. B. Cooper. Distinguishing the Arithmetical Hierarchy. *Preprint*, 1974.

[Coo04] S. B. Cooper. *Computability Theory*. Chapman and Hall/CRC Mathematics, 2004.

[Eps75] R. L. Epstein. Minimal Degrees of Unsolvability and the Full Approximation Construction. *Memoirs of the Am. Math. Soc.*, 3(162), 1975.

[Fef65] S. Feferman. Some Applications of the Notions of Forcing and Genericity. *Fundamenta Mathematicae*, 56:325–345, 1965.

[Fri57] R. M. Friedberg. Two Recursively Enumerable Sets of Incomparable Degrees of Unsolvability (Solution of Post's Problem, 1944). *Proc. Nat. Acad. Sci. U.S.A.*, 43(2):236–238, 1957.

[Ish03] S. Ishmukhametov. On a Problem of Cooper and Epstein. *J. Sym. Logic*, 68:52 – 64, 2003.

[Joc77a] C.G. Jockusch. Degrees of Generic Sets. In F. R. Drake and S. S. Wainer, editors, *Proceedings of Logic Colloquium*, Leeds, 1977.

[Joc77b] C.G. Jockusch. Simple Proofs of some Theorems of High Degrees. *Canad. J. Math.*, 29:1072 – 1080, 1977.

[JP78] C. G. Jockusch and D. B. Posner. Double Jumps of Minimal Degrees. *J. Sym. Logic*, 43:715–724, 1978.

[JS83] C.G. Jockusch and R. A. Shore. Pseudojump Operators I: The R.E. Case. *Trans. Am. Math. Soc.*, 275(2):599 – 609, 1983.

[JS84] C.G. Jockusch and R. A. Shore. Pseudojump Operators II: Transfinite Iterations, Hierarchies and Minimal Covers. *J. Sym. Logic*, 49(4):1205 – 1236, 1984.

[KP54] S. C. Kleene and E. L. Post. The Upper Semi-Lattice of Degrees of Recursive Unsolvability. *Ann. of Math.*, 59(2):379–407, 1954.

[Kum93] M. Kumabe. Generic Degrees are Complimented. *Ann. of Pure Appl. Logic*, 59(2):257–272, 1993.

[Ler83] M. Lerman. *Degrees of Unsolvability*. Springer, 1983.

[Lew] A. E. M. Lewis. The Search for Natural Definability in the Turing Degrees. *submitted.*

[Muc56] A. A. Muchnik. On the Unsolvability of the Problem of Reducibility in the Theory of Algorithms. *Dokl. Akad. SSSR (N.S)*, 108:194–197, 1956.

[Nie09] A. Nies. *Computability and Randomness.* Oxford University Press, 2009.

[NSS98] A. Nies, R. A. Shore, and T. A. Slaman. Interpretability and Definability in the Recursively Enumerable Degrees. *Proc. London Math. Soc.*, 77:241 – 291, 1998.

[Odi92] P. Odifreddi. *Classical Recursion Theory*, volume 1. Elsevier, 1992.

[Odi99] P. Odifreddi. *Classical Recursion Theory*, volume 2. Elsevier, 1999.

[Pos44] E. L. Post. Recursively Enumerable Sets of Positive Integers and their Decision Problems. *Bull. Amer. Math. Soc.*, 50:284–316, 1944.

[Pos79] D.B. Posner. A Survey of non-r.e. degrees $\leq 0'$. In F.R.Drake and S.S.Wainer, editors, *Proceedings of Logic Colloquium*, Leeds, 1979.

[Pos81] D. Posner. The Upper Semi-Lattices of Degrees $\leq \mathbf{0}'$ is complemented. *J. Sym. Logic*, 46:705–713, 1981.

[RT99] R.A.Shore and T.A.Slaman. Defining the Turing Jump. *Math. Res. Lett.*, 6:711–722, 1999.

[Sac61] G. E. Sacks. A minimal Degree less than $0'$. *Bull. Am. Math. Soc.*, 67:416 – 419, 1961.

[Sac63] G. E. Sacks. Recursive Enumerability and the Jump Operator. *Trans. Am. Math. Soc.*, 108:223 –239, 1963.

[Sho66] J. R. Shoenfield. A Theorem on Minimal Degrees. *J. Sym. Logic*, 31:539 – 544, 1966.

[Soa74] R. I. Soare. Automorphisms of the Lattice of Recursively Enumerable Sets. *Bul. Am. Math. Soc.*, 80:53 – 58, 1974.

[Soa87] R. I. Soare. *Recursively Enumerable Sets and Degrees*. Springer-Verlag, 1987.

[Spe56] C. Spector. On Degees of Recursive Unsolvability. *Ann. of Math.*, 64(3):581 – 592, 1956.

[SS99] R. A. Shore and T. A. Slaman. Defining the Turing Jump. *Math. Res. Lett.*, 6:711 – 722, 1999.

[Tur36] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 42:230–265, 1936.

[Tur39] A. M. Turing. Systems of Logic Based on Ordinals. *Proc. Lond. Math. Soc.*, 45:161–228, 1939.

[Yat70] C. E. M. Yates. Initial Segments of the Degrees of Unsolvability, part II: Minimal Degrees. *J. Sym. Logic*, 35:243 – 266, 1970.