Title
Indivisible Characteristics of Recursively Enumerable Sets

Author
Chih, Ellen S.

Publication Date
2015

Peer reviewed|Thesis/dissertation

Indivisible Characteristics of Recursively Enumerable Sets

by

Ellen S. Chih

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Leo A. Harrington, Co-chair
Professor Theodore A. Slaman, Co-chair
Associate Professor Lara Buchak

Spring 2015

Indivisible Characteristics of Recursively Enumerable Sets

Abstract

Indivisible Characteristics of Recursively Enumerable Sets

by

Ellen S. Chih

Doctor of Philosophy in Mathematics

University of California, Berkeley

Professor Leo A. Harrington, Co-chair

Professor Theodore A. Slaman, Co-chair

A split of an r.e. set A is a pair of disjoint r.e. sets whose union is A. We investigate information theoretic properties of r.e. sets and properties of their enumerations, and whether these properties are preserved under splittings. The first part is involved with dynamic notions. An r.e. set is speedable if for every computable function, there exists a finite algorithm enumerating membership faster, by the desired computable factor, on infinitely many integers. Remmel [5] (1986) asked whether every speedable set could be split into two speedable sets. Jahn [7] published a proof, answering their question positively. However, his proof was seen to be incorrect and we construct a speedable set that cannot be split into speedable sets, answering their question negatively. We also prove additional splitting results related to speedability.

The second part is involved with effectively closed sets. We introduce the notion of being recursively avoiding and prove several results.

To my father

Contents

Contents **ii**

List of Figures **iii**

1 Introduction **1**
1.1 Overview 1
1.2 Technical conventions and definitions 2

2 On the weak jump and speedability **3**
2.1 Introduction 3
2.2 The main question 4
2.3 Notation and splittings 5
2.4 One split 6
2.5 Two splits 18
2.6 The general case 25
2.7 Further generalizations and questions 33

3 On semilow$_2$ sets **35**
3.1 Overview 35
3.2 Requirements 35
3.3 Priority Tree 36
3.4 Notation 36
3.5 Intuition 37
3.6 Construction 37
3.7 Verification 39

4 Recursively avoiding reals **42**
4.1 Introduction 42
4.2 Existence of rec avoiding reals in every Δ^0_2 degree 43

Bibliography **46**

List of Figures

2.1 The dynamics between the Q- and S-strategies. 10
2.2 A table summarizing the possible switching that can occur for the two split case. The "Value of Z_0 (or Z_1)" for configuration n refers to the value of Z_0^ν (Z_1^ν) for nodes ν between the node for configuration n and the node for configuration $n+1$. An equation in R_{node} for configuration n refers to the equation in R_η where η is the value of the node for configuration n. 20
2.3 In this figure, we assume that there is no node with true outcome c above γ except for β, α and δ. Assuming that this figure signifies a portion of the true path, the dashed edges between nodes represents the true path in between the two nodes. 20
2.4 Assuming that this figure signifies a portion of the true path, a dotted line signifies that the true path does not go this way and the dashed edges between nodes signifies the true path in between the two nodes. Here, we assume that all Q-nodes on the true path between ν and γ that work on the same Q_h requirement as α has true outcome c. Technically, there are other nodes working on S-requirements between α and δ and δ and ν but we have not included them in the picture. . . 27

Acknowledgments

First, I would like to thank my advisors, Leo Harrington and Theodore Slaman for their guidance, valuable advice, kindness, encouragement, patience and insightful discussions. They suggested many interesting problems and questions for me to think about and were always willing to share their wisdom. Learning from them was a very enjoyable experience. I am very grateful that I had the chance to learn from both of them.

Secondly, I would like to thank Lara Buchak for being on my dissertation committee and for her valuable comments on my dissertation. I would also like to thank Rod Downey for suggesting problems and in particular, the main question of Chapter 2 to work on.

Thirdly, I would like to thank Theodore Slaman and the Department of Mathematics for financially supporting me during graduate school. I would also like to thank Barb Waller and Marsha Snow for all of their help.

Finally, I would like to thank my family and especially my father. Without his encouragement and support, graduate school would not have been possible for me. Whenever I had doubts, he always insisted for me to continue forward and put my doubts behind.

Chapter 1

Introduction

1.1 Overview

When one investigates the recursive properties of a recursive enumerable (r.e.) set A, one considers the aggregate matter of A instead of the exact numbers. If one does a recursive permutation of a set, the resulting set retains the same recursion theoretic properties. Similarly, if one removes a recursive set from A, there is a negligible difference between A and the resulting set. However, when one removes a nonrecursive set, things are not as clear. Are there intrinsic properties of A that are lost when one removes a nonrecursive set? What recursion theoretic properties are still retained?

These questions lead to the study of splittings of an r.e. set. A splitting of an r.e. set A is a pair of disjoint r.e. sets A_0, A_1 whose union is A. We say that a splitting of A is nontrivial if both A_0, A_1 are not recursive. In this work, we investigate information theoretic properties of r.e. sets and properties of their enumerations, and whether these properties are atomic; that is, whether there is a r.e. set with this property such that its elements cannot be split into two r.e. sets which have the same property.

The first part is involved with dynamic notions. An r.e. set is speedable if for every computable function, there exists a finite algorithm enumerating membership faster, by the desired computable factor, on infinitely many integers. Being speedable is closely related to the weak jump. While working with the universal splitting property, Remmel asked whether every speedable set could be split into two speedable sets. Jahn published a proof, answering their question positively. However, his proof was seen to be incorrect. We construct a speedable set that cannot be split into speedable sets, answering their question negatively. We also prove other results on the aspects of enumeration and on dynamic notions.

The second part is involved with Π^0_1 classes. We introduce the notion of rec avoiding reals and prove several results relating to them. In particular, we give an alternative proof of the folklore fact that if A is r.e. and not recursive, there exists a set X of the same degree as A such that X is not contained in any countable Π^0_1 class.

1.2 Technical conventions and definitions

The proofs in the first two parts are priority arguments done by constructions on a priority tree. We list conventions and definitions that we use throughout the proofs. When we refer to Theorem n or Lemma n in Chapter k, we are referring to Theorem $k.n$ or Lemma $k.n$.

Our notation follows [11]. We use the following conventions.

Unless specified otherwise, all sets are assumed to be recursively enumerable (r.e.). Fix the universal standard enumeration of r.e. sets as defined in [11]. Let W_i denote the i^{th} r.e. set in this enumeration. Let $W_{i,s}$ denote the subset of W_i as enumerated at the end of stage s. Let $\Phi_i(x)$ denote the stage s when x enters W_i (i.e. the least s such that $x \in W_{i,s}$). We sometimes refer to Φ_i as Φ_V where $V = W_i$. For α, β that are nodes on our priority tree $T = \Lambda^{<\omega}$, we say that α is to the left of β, written $\alpha <_L \beta$, if there is some $\gamma \in T$ and $a, b \in \Lambda$ such that $a < b$ and $\gamma^\frown\langle a\rangle \subseteq \alpha$ and $\gamma^\frown\langle b\rangle \subseteq \beta$ (where $<$ is the ordering on Λ and $^\frown$ denotes concatenation). By the true path, we mean the leftmost path travelled through infinitely often in the construction. By "true" outcome o of α, we mean that o is the outcome of α on the true path. We call a node working on an S-strategy *active* at stage s if the node has acted at some stage $t < s$ and has not been cancelled (or reset).

Chapter 2

On the weak jump and speedability

2.1 Introduction

Given a property P of r.e. sets, we can ask whether P is preserved under splittings. One way to formulate the question is the following:

Question 2.1.1 (Is P divisible)**.** *For all A with property P, is there a split A_0, A_1 such that both A_0 and A_1 have property P?*

By the Friedberg Splitting Theorem, every nonrecursive set can be split into two nonrecursive sets and thus being not recursive is a divisible property. Another example of a divisible property is the property of being not K-trivial. This fact is folklore but as we do not have a reference for it, we provide a short proof for completeness. [1]

Lemma 2.1.2. *For every A that is not K-trivial, there is a split A_0, A_1 such that A_0 is not K-trivial and A_1 is not K-trivial.*

Proof. To see that we can build such a split, we utilize the Sacks preservation strategy to satisfy the following requirements for constants c:

$$R_{i,c} : (\exists n) K(A_i \upharpoonright n) > K(n) + c$$

Let the length of agreement $l_s(i,c)$ for requirement $R_{i,c}$ at stage s be is the greatest l such that $K(A_i \upharpoonright l) \leq K(l) + c$ and let $r_s(i,c)$ be the max of $l_t(i,c)$ for $t \leq s$. We say that a requirement $R_{i,c}$ is injured if some $x < r_s i, c$ enters A_i.

At each stage s, we mirror the proof of the Sacks Splitting Theorem so that if x enters A at stage s, we determine the highest priority $R_{i,c}$ that would be injured if x goes in and put x into A_{1-i}. If no such requirement exists, we put x into A_0.

[1] An interesting sidenote to make is the following. For any A not K-trivial and any split A_0, A_1 of A, we can immediately conclude that at least one of A_0 or A_1 is not K-trivial as the join of two K-trivials is still K-trivial.

To see that this construction works, assume otherwise and assume without loss of generality A_i is the set that is K-trivial. Let c be the least such that $(\forall n)K(A_i \upharpoonright n) \leq K(n) + c$. By definition of the length of agreement, we know that the limit of $l_s(i, c)$ goes to infinity. As c was the least, we know that eventually $R_{i,c}$ stops being injured after some stage s. We now argue that A_{1-i} is computable. Fix some m. Let $t > s$ be such that $l_t(i, c) > m$. As $R_{i,c}$ is not injured after stage s, $A_{1-i} \upharpoonright m$ at stage t is equal to $A_{1-i} \upharpoonright m$. As m was arbitrary, A_{1-i} is computable and thus K-trivial. However, we assumed that A_i is K-trivial and the join of two K-trivials is K-trivial so A is K-trivial, contradicting our assumption that A was not K-trivial. □

What about examples of properties that are not divisible? By Downey and Shore's [4] result of the existence of a high set whose nontrivial splits are low, being high is an example of a property that is not divisible. In this chapter, we introduce another property that is not divisible, relating to the weak jump ($\{e : \overline{A} \cap W_e \neq \emptyset\}$) and see how different measuring complexity by the weak jump is compared to measuring complexity by the jump. In the next chapter, we also introduce another property that is not divisible.

2.2 The main question

[2] An r.e. set A is *speedable* if for every recursive function, there exists a program enumerating membership in A faster, by the desired recursive factor, on infinitely many integers. Blum introduced speedable sets in 1967 and Blum and Marques later expanded the notion of speedability to r.e. sets. According to Blum and Marques [2], "[a]n important goal of complexity theory ... is to characterize those partial recursive functions and recursively enumerable sets having some given complexity properties, and to do so in terms which do not involve the notion of complexity" and speedability was a step in the direction of this goal.

Soare [10] gave an "information theoretic" characterization of speedable sets in terms of a well-studied class of r.e. sets. He proved that a set A is speedable if and only if A is not *semi-low*, namely,

$$\{e : \overline{A} \cap W_e \neq \emptyset\} \not\leq_T \emptyset'$$

where W_e denotes the e^{th} r.e. set. The set $\{e : \overline{A} \cap W_e \neq \emptyset\}$ is called the *weak jump* of A.

In 1986, Remmel [5] asked whether every speedable set could be split into speedable sets. This question is the main topic of this paper.

Question 2.2.1 (The main question of Chapter 1)**.** *[5] Can every speedable set be split into two speedable sets?*

The answer was thought to be positive for some time. In 1993, Downey, Jockusch, Lerman and Stob [6] proved that every hyper-hyper-simple set can be split into speedable

[2] The following work is to appear in the *Journal of Symbolic Logic.*

sets, contributing evidence for a positive answer. In 1999, the question was thought to be resolved when Jahn [7] published a proof that every speedable set could be split into two speedable sets. His paper was cited [9] as a positive case for other splittings of sets with related complexity properties and cited again [3] for introducing various complexity properties and splittings. However, Downey [private communication] pointed out that the proof in [7] is incorrect.

The main goal of this paper is to construct a speedable set that cannot be split into speedable sets (i.e. to negatively answer Question 1.1). The proof is by a tree construction but there are infinite positive requirements to the left of the true path. In most tree constructions, nodes to the left of the true path are guessing a Π^0_2 outcome that is seen to be false so the action to the left settles down and becomes finite. However, in our construction, "settling down" means an infinite positive Π^0_1 action, namely, all x (of some particular type) get put into our set quickly.

The first section dealing with the main question of this chapter sets notation and introduces the main theorem of the paper. The second section examines the case where we are dealing with only one split and the third examines the case where we are dealing with two splits. The proof of the main theorem is given in the fourth section. In the final section, we examine various ways the main theorem could be improved.

2.3 Notation and splittings

Our notation follows [11]. We use the following conventions.

Unless specified otherwise, all sets are assumed to be recursively enumerable (r.e.). Fix the universal standard enumeration of r.e. sets as defined in [11]. Let W_i denote the i^{th} r.e. set in this enumeration. Let $W_{i,s}$ denote the subset of W_i as enumerated at the end of stage s. Let $\Phi_i(x)$ denote the stage s when x enters W_i (i.e. the least s such that $x \in W_{i,s}$). We sometimes refer to Φ_i as Φ_V where $V = W_i$. For α, β that are nodes on our priority tree $T = \Lambda^{<\omega}$, we say that α is to the left of β, written $\alpha <_L \beta$, if there is some $\gamma \in T$ and $a, b \in \Lambda$ such that $a < b$ and $\gamma^\frown\langle a\rangle \subseteq \alpha$ and $\gamma^\frown\langle b\rangle \subseteq \beta$ (where $<$ is the ordering on Λ and $^\frown$ denotes concatenation). By the true path, we mean the leftmost path travelled through infinitely often in the construction. By "true" outcome o of α, we mean that o is the outcome of α on the true path.

Definition 2.3.1. Let A be an r.e. set. A is *nonspeedable* if and only if there exists some i such that $W_i = A$ and a recursive function h such that for all j,

$$W_j = A \Rightarrow (a.e.x)[x \in A \Rightarrow \Phi_i(x) \leq h(x, \Phi_j(x))]$$

where $(a.e.x)$ is all x mod finite.

A set A is *speedable* if and only if it is not nonspeedable.

By [10], instead of just one enumeration being "optimal", being nonspeedable also implies that every enumeration is optimal. We use this equivalent condition interchangeably:

Definition 2.3.2. A is *nonspeedable* if and only if for every i such that $W_i = A$ there exists a recursive function h such that for all j,

$$W_j \subseteq^* A \Rightarrow (a.e.x)[x \in A \Rightarrow \Phi_i(x) \leq h(x, \Phi_j(x))] \qquad (*)$$

where $\subseteq^*$ is subset mod finite.

A *splitting* of an r.e. set B is a pair of disjoint r.e. sets X, Y whose union is B.

Theorem 2.3.3. *There is a speedable set B such that if X and Y form a split of B, at least one of X or Y is nonspeedable.*

By [10], being nonspeedable is equivalent to being semilow and the following corollary follows immediately:

Corollary 2.3.4. *There is a non-semilow set B such that if X and Y form a split of B, at least one of X or Y is semilow.*

To construct $B = W_i$ to be speedable, we diagonalize against all recursive functions h that could witness $(*)$; i.e. for each recursive h, we build an r.e. $W_j \subseteq^* B$ that witnesses the failure of $(*)$ for h.

To ensure that B cannot be split into speedable sets, if X and Y form a split of B, we first try to ensure that X is nonspeedable. This attempt may interfere with our making B speedable, in which case we see that we have the means to ensure that Y is recursive.

2.4 One split

We first deal with the case where we are given a split of $B = W_e$ recursively in e, i.e. W_i, W_j are splits of B given by a recursive function f such that $f(e) = (i, j)$.

Lemma 2.4.1. *For every recursive function f, there is a speedable set B such that if X, Y is the split of B given by f, at least one of X or Y is nonspeedable.*

Proof. We recursively enumerate B and ensure that B is speedable.

The proof is a tree construction in the sense of [11] (Chapter 14). Nodes work on strategies. For a node α working on a strategy, α builds B_α which is intended to be an enumeration of B with a different timescale. $B_\emptyset$ is B. α also builds P_α, a recursive set which will be used as a pool of numbers. We refer to P_α as the pool lower priority nodes use. α uses witnesses from the pool given by the previous node. $P_\emptyset$ is ω. A member in a pool can become "used" but does not necessarily have to go into B. One way a number x can become used is if x enters B_α. If a number in α's pool is "unused", α can keep it out of B.

Modulo finite injury, a node can recursively tell which pool it is using. A node can also recursively tell which elements in a pool it can keep out.

By the fixed point theorem, we have some index k such that $B = W_k$. We technically want to show that W_k is speedable.

Requirements

To build a speedable set, we build an r.e. set B satisfying the following requirements, one for each partial recursive function h:

$$Q_h^* : (\exists M \subseteq^* B)(\exists^\infty x)(\Phi_B(x) > h(x, \Phi_M(x)),$$

where $\Phi_B(x)$ is according to when we put x into B.

As there is at most a recursive difference between when x enters B and when x enters W_k, satisfying Q_h^* shows that W_k is speedable.

We do not satisfy Q_h^* directly. Instead, we satisfy the following:

$$Q_h : (\exists M \subseteq^* B)(\exists^\infty x)(\Phi_B(x) > h(x, \Phi_M(x)) \vee Y \text{ is recursive.}$$

We work on Q_h instead of Q_h^* as we are unable to achieve the first disjunct if X is speedable. If Y is recursive, we have another node working on Q_h.

We try to build an r.e. set $M \subseteq^* B$ satisfying $(\exists^\infty x)(\Phi_B(x) > h(x, \Phi_M(x))$ by putting x's into M and keeping them out of B until stage $h(x, \Phi_M(x)) + 1$. If we succeed for only finitely many x's, we conclude that Y is recursive and thus nonspeedable.

To make one of X or Y nonspeedable, we either make Y recursive by some Q-strategy or we make X nonspeedable by satisfying the following requirements for the recursive g defined below and all r.e. sets V (an equivalent to Definition 2.1):

$$S_V : (\exists x)(x \in V \wedge x \notin X) \text{ or } (\forall x)[x \in V \Rightarrow \Phi_X(x) \leq g(x, \Phi_V(x))]$$

Define $g(x, s)$ to be the least stage t that x enters X or Y if x is in B_s. Otherwise, we let $g(x, s)$ equal 0. Observe that g is total and recursive.

We also have the following requirements to help the Q-strategies (see Lemma 3.8 for where the D_i requirements are required in the verification):

$$D_i : \Delta_i \neq B$$

Remark

It may seem redundant to have the D strategies when every speedable set is not recursive but showing that the set we build, B, is speedable requires showing that a particular recursive description does not work. Satisfying all of the D strategies shows that B is not recursive and thus the particular recursive description does not work. It is true that we could incorporate this into the Q strategy but we have chosen to separate the strategies in order to not overly complicate the construction.

Outcomes of Q-nodes

A node α working on a Q-strategy has outcomes: ∞ (for infinitely many), c (for cofinitely many) and h. Along the ∞ outcome, infinitely many x's in M_α enter B after stage $h(x, \Phi_{M_\alpha}(x))+1$. Along the c outcome, cofinitely many x's in M_α enter B before $h(x, \Phi_{M_\alpha}(x))+1$ and we see that we have the means to ensure that Y is recursive. Along the h outcome, h is partial.

Outcomes of S-nodes

A node α working on an S-strategy has outcomes: k (for keep out of B) and s (for ($\sharp$) where ($\sharp$) will be defined in the description of the strategy). Along the k outcome, either there is some x in both V and Y or α can keep some element in V out of B. Along the s outcome, α does not achieve $(\exists x)(x \in V \wedge x \notin X)$ and α tries to achieve $x \in X \Rightarrow \Phi_X(x) \leq g(x, \Phi_V(x))$ for all x.

Outcomes of D-nodes

A node α working on a D-strategy has outcomes: a (for act) and d (for diverge). Along the a outcome, $\Delta_i(x)$ converges and equals 0 and α would like to put x into B. Along the d outcome, $\Delta_i(x)$ diverges or $\Delta_i(x)$ converges and does not equal 0 and α tries to keep x out of B.

Priority Tree

Fix a recursive ordering of the Q- and S-requirements.

Let $\Lambda = \{\infty, c, h, k, s, a, d\}$ with ordering $\infty < c < h < k < s < a < d$. The tree is a subset of $\Lambda^{<\omega}$ and is built by recursion as follows:

Assign the highest priority Q_h requirement to the empty node and let ∞, c and h be its successors. Assume that we assigned a requirement to $\beta = \alpha \upharpoonright (|\alpha| - 1)$, which we call the β-requirement. We now assign a requirement to α. If c is the last node of α, assign β-requirement to α and let its successors be ∞ and h.

Suppose c appears in α and is not the last node of β: If the β-requirement is a D-requirement, assign the highest priority Q-requirement that has not been assigned so far and let its successors be ∞ and h. If the β-requirement is a Q-requirement, assign the highest priority D-requirement that has not been assigned so far and let its successors be a and d.

Suppose c does not appear in α: If the β-requirement is a Q-requirement, assign the highest priority D-requirement that has not been assigned so far and let its successors be a and d. If the β-requirement is a D-requirement, assign the highest priority S-requirement that has not been assigned so far and let its successors be k and s. Otherwise, assign the highest priority Q-requirement that has not been assigned so far and let its successors be ∞, c and h.

Observe that we only allow ∞ and h to be successors for Q-requirement nodes that appear after an instance of c. We will prove in the verification (Lemma 3.8) that on the true path, if the c outcome occurs, all Q_h-nodes α after it must achieve: $(\exists^\infty x)(\Phi_B(x) > h(x, \Phi_{M_\alpha}(x))+1)$ for M_α that α builds if h is not partial.

Dynamics of the construction

We now describe exactly how the Q_h nodes α and the S nodes achieve their goals. At a typical stage of the construction, we will need to consider the configurations as indicated by Figure 1 below. At α, Q_h will attempt to pick a witness, wait for a delay determined by h, and put the element into its local version of B, giving this to β for it to process in a like fashion, assuming that β is a $Q_{h'}$ node (strictly speaking, this is likely a $\beta' \subset \beta$). Eventually all such Q nodes process x and it will enter B. We refer to this as x entering B the *slow* way.

Now, it could be that an S-node might care about x and interrupt this procedure. There are two ways this can occur. First some S node ν of higher priority than α discovers $x \in V_\nu[s]$. Then, ν can see a global win and will restrain x with priority ν. We will allow this to happen even if ν is strictly left of the current approximation to the true path. An inductive argument will show that such restrains with a ν-node injuring the action of α with $\nu <_L \alpha$ can happen at most finitely often.

The most important new idea in this construction is that other S nodes can affect the action of α. We will allow η nodes for S_{V_i} for $\alpha^\frown\langle\infty\rangle \subseteq \eta$ to also pull witnesses from α. This will only be allowed if such a node η is *active*, meaning that we have actually visited it at some stage $s' < s$. If these η nodes prevent α from satisfying the first conjunct of Q_h (i.e. cofinitely many x's go into B before stage $h(x, \Phi_{M_\alpha}(x)) + 1$), we can give a proof that Y is recursive. The construction does the following. If an active η sees such an x then η *immediately* puts x into B and α makes sure that the next such x must be large. Notice that the hypothesis of the S node being correct implies that x must enter X and not Y. Thus, assuming that the c outcome is the true outcome for α, no small number can enter Y since we can ensure that pulled numbers that enter B are from a set that is disjoint from Y.

The point is the following. The only place (small) numbers that enter B will come from will be from those nodes extending $\alpha^\frown\langle c\rangle$. If η above saw these numbers *before* they reached α in their upward climb, then η would simply restrain them and win forever. Hence they can only be pulled by η after they reach α. The relevant x always enters X. Then we will reset the sizes of numbers so that we get a recursive description of Y.

The reader should note that if $\alpha^\frown\langle\infty\rangle$ is *not* on the true path, then we will only visit η's extending $\alpha^\frown\langle\infty\rangle$ finitely often so the true outcome of α is c if h is total. This outcome has a version of Q_h attached which will succeed in meeting h since we *know* that such active $\eta \supseteq \alpha^\frown\langle\infty\rangle$ cannot pull the element *before* they get to α.

It is important to note that there are no S requirements below the c outcome since if c is the true outcome, we have a proof that Y is recursive. If α is on the true path and c is the outcome of α on the true path, Π_2 (in Figure 1) reflects the fact that infinitely often, we

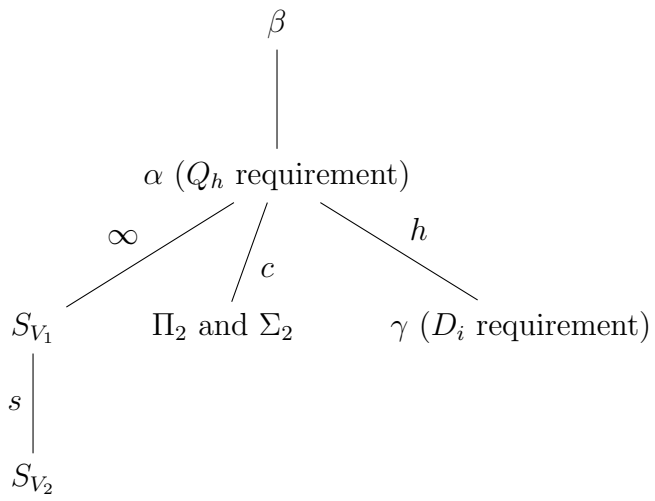

Figure 2.1: The dynamics between the Q- and S-strategies.

assign numbers to go into M_α and Σ_2 (in Figure 1) reflects the fact that only finitely many numbers stay in M_α long enough to see h defined.

As an S-strategy can become infinite positive, the Q- and D-strategies cannot be infinite negative. Furthermore, the Q- and D-strategies cannot have negative restraints that effect S-strategies. Negative restraints that go up and drop down, such as in the minimal pair construction, are also not allowed.

We now turn to the formal details.

Convention

We use the following convention for labeling an element "used" or "unused" at α and at stage s. If an element becomes used, it is used forever. If an element is picked by a D-strategy, it is used. If an element is picked by an S-strategy, it is used. If an element goes into B_α, it is used. For a Q-strategy, a picked element is still "unused" (see step (1) of the strategy for Q_h) but becomes used when put into M_α or into B_α (see step 2(b) of the strategy for Q_h).

This convention is needed in the verification (see Lemma 3.4) to prove that for every node α, if an element x is unused and in α's pool, x does not go into B while α restrains x.

In the following strategies, we work on a node α whose predecessor is β. α uses the pool given by its predecessor, denoted by P_β. Recall that the pool of the empty set is ω and recall that $B_\emptyset = B$.

Strategy for D_i

Takes parameters: (B_β, P_β). Outputs parameters $(B_\alpha, P_\alpha) = (B_\beta, P_\beta \setminus x_\alpha)$.

Pick an unused witness $x_\alpha \in P_\beta$. Keep x_α out of B and wait for $\Delta_i(x_\alpha)$ to converge to 0. If it does, put x_α into B_β.

If an element x enters B_α, put x into B_β unless D_i is assigned to the h outcome, in which case we put x into B_η (where η is β's predecessor). Define P_α to be $P_\beta \setminus x_\alpha$ and $B_\alpha = B_\beta$.

Note that x_α may have to enter other sets before entering B.

Strategy for S_V

Takes parameters: (B_β, P_β). Outputs parameters $(B_\alpha, P_\alpha) = (B_\beta, P_\beta \setminus x_\alpha)$ or $= (B_\beta, P_\beta)$.

Below, we go through steps (1) - (4) to find a witness x_α in V that can be permanently be kept out of X either by action of α, by action of a higher priority node or by the existence of an x already in $V \cap Y$. Step (3)(b) is allows an S-node to grab and restrain any element x before x reaches a higher priority node in its upward climb. Step (3)(b) also stops such x, if chosen, from continuing in its upward climb.

While x_α has not been defined, go through the following steps in order. Upon defining x_α, continue to define B_α and P_α as below.

1. Ask whether there is some x in $V \cap Y$. If so, let x_α be the least such x and continue to define P_α and B_α as below. Note that S_V is satisfied as $V \not\subseteq X$.

2. Ask whether there is a γ working on an S-strategy such that x_γ is defined and $x_\gamma \in V$. If so, let $x_\alpha = x_\gamma$ and restrain lower priority requirements from putting x_α into B. Note that if α permanently restrains x_α, S_V is satisfied as x_α witnesses $(\exists x)(x \in V \wedge x \notin B)$, hence $V \not\subseteq X$.

3. Look for a γ working on a D-strategy with $x_\gamma \in V$ as follows.

   a) Ask whether there is a higher priority γ already restraining $x_\gamma \in V$. If so, let $x_\alpha = x_\gamma$. Note that if γ permanently restrains $x_\gamma = x_\alpha$, then S_V is satisfied as x_α witnesses $(\exists x)(x \in V \wedge x \notin B)$ hence $V \not\subseteq X$ as in case (2). If γ is reset, we reset x_α as well.

   b) Ask whether there is some weaker priority γ such that x_γ is not in B and not in B_δ for any δ of higher priority than α. If so, let $x_\alpha = x_\gamma$ and restrain lower priority nodes η from putting x_α into B_ξ where ξ is η's predecessor. Reset the γ node (so that if we travel to the γ node again, it chooses another witness). Note that if α permanently restrains x_α, S_V is satisfied as x_α witnesses $(\exists x)(x \in V \wedge x \notin B)$, hence $V \not\subseteq X$ as in case (2).

4. Ask whether there is an unused $x \in V \cap P_\beta$. If so, let x_α be the least such x and restrain lower priority requirements from putting x_α into B. Note that if α permanently restrains x_α, S_V is satisfied as x_α witnesses $(\exists x)(x \in V \wedge x \notin B)$, hence $V \not\subseteq X$ as in case (2).

While (1) - (4) do not hold, commit to:

$$\text{When } x \text{ enters } V, \text{ put } x \text{ into } B \text{ immediately.} \qquad (\sharp)$$

With ($\sharp$), if x enters V at stage s, it must enter B at stage s so $\Phi_X(x) = g(x, \Phi_V(x))$. Thus, S_V is satisfied (by satisfaction of its second disjunct).

If an element enters B_α, put it into B_β. If x_α is undefined, let $P_\alpha = P_\beta$. Otherwise, let $P_\alpha = P_\beta \setminus x_\alpha$. In either case, let $B_\alpha = B_\beta$.

Strategy for Q_h

Takes parameters: (B_β, P_β). Builds parameters: R_α, M_α, P_α^∞, B_α^c. Outputs parameters

$$(B_\alpha, P_\alpha) = \begin{cases} (B_\beta, P_\alpha^\infty) & \text{if outcome is } \infty \\ (B_\alpha^c, R_\alpha) & \text{if outcome is } c \\ (B_\beta, R_\alpha) & \text{if outcome is } h \end{cases}$$

To explain, R_α is a recursive set, B_α^c is an r.e. set and P_α^∞ is a recursive set.

Recall (see section, Dynamics of the construction) that in the c outcome, we need to ensure that Y is a recursive set. To achieve this, we ensure that Y is contained in a recursive subset of B (namely, $B \setminus R_\alpha$) so Y is part of a split of a recursive set and thus Y is recursive.

Also recall that in the c outcome, pulled numbers have to enter X and such (small) numbers have to come from nodes below $\alpha^\frown\langle c\rangle$ and reach α in their upward climb. The role R_α plays is to provide a pool of elements for these numbers so pulled numbers only come from R_α (but do not have to be contained in a version of R_α that has been reset). Furthermore, we ensure that $B \cap R_\alpha$ only consists of these pulled numbers, which enter X in the c outcome. Thus in the c outcome, $Y \subseteq B \setminus R_\alpha$ and we will ensure that $B \setminus R_\alpha$ is recursive.

In order to ensure that $B \cap R_\alpha$ only consists of pulled numbers in the c outcome, R_α can be reset as numbers that are not pulled and enter B cannot be in R_α. In the ∞ outcome, R_α is reset infinitely many times.

The role of P_α^∞ is the following: As the construction tries to ensure that $B \setminus R_\alpha$ is recursive and can potentially reset R_α infinitely often, we need a pool of elements for the ∞ outcome, namely, P_α^∞. P_α^∞ will only increase when we see another witness for the ∞ outcome so as long as we ensure that $(B \setminus R_\alpha) \cap \overline{P_\alpha^\infty}$ is recursive, $B \setminus R_\alpha$ is recursive in the c outcome as P_α^∞ is finite. In the ∞ outcome, P_α^∞ will be recursive and infinite.

Now, we explain the intuition behind steps (1) - (4) below: Step (1) builds R_α and step (2) builds M_α.

Step (3) looks at the case where x has entered M_α (i.e. x has reached α in its upward climb) and no node has pulled x. In (3)(a), we put x into B_β to let x continue in its upward climb. In 3(b), we add another element to P_α^∞ as we see another instance of $(\exists^\infty x)(\Phi_B(x) > h(x, \Phi_{M_\alpha}(x)) + 1)$. We also reset R_α as x is not a pulled number but is in R_α and we only

want $B \cap R_\alpha$ to contain pulled numbers. We put all elements in M_α into B at this step as these elements are no longer in R_α after the reset.

In step (4), we put all unused elements below x that are in $\overline{R_\alpha} \cap \overline{P_\alpha^\infty}$ into B. This step forces later witnesses to be large and in particular, bigger than x.

Now we formally state steps (1) - (4):

1. Pick an unused witness $x \in P_\beta$. Put x into R_α. x is subject to steps (2) and (3). Note that x does not necessarily have to enter M_α.
2. While x is not in M_α,
   a) Restrain x from entering M_α until it enters B_α^c (by the action of lower priority nodes).
   b) If x goes into B_α^c, put x into M_α.
3. If x enters M_α,
   a) If the current stage is greater than $h(x, \Phi_{M_\alpha}(x)) + 1$ and x has not entered B_β, put x into B_β.
   b) If x is successfully is kept out of B until stage $h(x, \Phi_{M_\alpha}(x))+1$, we do the following: Pick an unused witness y in $P_\beta \cap \overline{R_\alpha}$. Put y in P_α^∞ and extend P_α^∞'s recursive description up to y, i.e. if an arbitrary $z \leq y$ is in P_α^∞ at the current stage, the description says z is in and if z is not in P_α^∞ at the current stage, the description says z is out. Reset R_α and put all elements in M_α into B (since these elements are no longer in our current R_α).
4. Put all unused elements in $\{y : y < x\} \cap \overline{R_\alpha} \cap \overline{P_\alpha^\infty}$ into B.

If we have the c outcome, $P_\alpha = R_\alpha$ and $B_\alpha = B_\alpha^c$. If we have the ∞ outcome, $P_\alpha = P_\alpha^\infty$ and $B_\alpha = B_\beta$.

Observe that we would like to achieve:

(†) If x enters M_α, x is kept out of B until stage $h(x, \Phi_{M_\alpha}(x)) + 1$.

If (†) is achieved for infinitely many x's, Q_h is satisfied. Recall we use ∞ outcome to denote that infinitely many x's achieve (†). Thus in the ∞ outcome, Q_h is satisfied.

Construction

We call a node working on an S-strategy *active* at stage s if the node has acted at some stage $t < s$ and has not been cancelled (or reset).

At stage s, define δ_s (an approximation to the true path) by recursion as follows. Suppose that $\delta_s \upharpoonright e$ has been defined for some $e < s$. Let α be the last node of $\delta_s \upharpoonright e$. We now define $\delta_s \upharpoonright (e+1) \supseteq \delta_s \upharpoonright e$.

1. If α is a Q-node, look to see if it is waiting for a number x to go into B_β (where β is α's predecessor), i.e. there is some x in M_α that has not entered B_β. If so, look to see if an active node η to to the left or below $\alpha^\frown\langle\infty\rangle$ would like to put x in (or keep x out if η is to the left). Let η act. If there are no remaining numbers that α is waiting on, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle c\rangle$. Otherwise, look to see if there is some z in the remaining numbers such that $h(z, \Phi_{M_\alpha}(z))$ has converged and our current stage is greater than $h(z, \Phi_{M_\alpha}(z)) + 1$. If so, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle\infty\rangle$. If not and some x was pulled, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle c\rangle$. Otherwise, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle h\rangle$.

2. If α is an S-node, look to see if x_α has been defined. If so, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle k\rangle$. Otherwise, go through steps (1) - (4) in the strategy for S_V. If one of (1) - (4) holds and we can define x_α, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle k\rangle$. Otherwise, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle s\rangle$.

3. If α is a D-node, look to see whether $\Delta_i(x_\alpha)$ has converged and equals 0. If so, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle a\rangle$. Otherwise, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle d\rangle$.

Reset nodes to the right of δ_s and let active S-nodes to the left of δ_s or below the ∞ outcome for one of the nodes in δ_s act. At substage $t \leq s$, let $\delta_s \upharpoonright t$ act according to its description in the strategies above, i.e. $\delta_s \upharpoonright t$ enumerates numbers in its sets and extends their definitions.

Verification

Recall that the true path denotes the leftmost path traveled through infinitely often by the construction. β is α's predecessor unless stated otherwise.

Lemma 2.4.2. *For α on the true path working on a D-requirement, we only reset x_α at most finitely many times. For α on the true path working on a Q-requirement, $M_\alpha \subseteq^* B$.*

Proof. Let α be a node on the true path. We travel to the left of α at most finitely often in the construction so in the limit, there are only finitely many higher priority active S-strategies. Now, consider only the stages after which we do not go to the left of α.

Suppose that α is working on a D-requirement. At any stage s, we only reset x_α due to a higher priority active S-strategy. We only reset x_α at most once per every given S-strategy due to 3(b) in the S strategy and thus we only reset x_α at most finitely many times.

Suppose that α is working on a Q-requirement. We only restrict x in M_α from entering B due to a higher priority active S-strategy. We only restrict at most one element per every given S-strategy due to 3(b) in the S strategy and thus $M_\alpha \subseteq^* B$. □

The next lemma shows that if an x has started on its upward climb, it either reaches B or is reset. The next lemma also shows that if x has started on its upward climb, this action must be initiated by a D-node.

Lemma 2.4.3. *If x_α is picked for a node α on the true path and α puts x_α into B_β, it is either reset or put into B. For $\alpha \neq \emptyset$, if x is put in B_α by a lower priority node, x was picked as a witness for a lower priority node γ working on a D-strategy.*

Proof. Suppose that x_α is never reset. Consider only the stages after which we do not go to the left of α. By assumption, x_α is put into B_β. We now prove on the true path that if x_α enters B_γ for $\gamma \subseteq \alpha$ then x_α enters B by an S-strategy or x_α enters B_δ where δ is γ's predecessor. Suppose that x_α enters B_γ for $\gamma \subset \alpha$. If x_α is grabbed by an active S-node to the left, it is put in B. If γ is working on a S-requirement, γ either puts x_α into B by ($\sharp$) or puts x_α into B_δ as we assumed that x_α is never reset. If γ is working on a D-requirement, γ puts x_α into B_δ. If γ is working on a Q-requirement, either x was put into B_δ as $B_\delta = B_\gamma$ if α extends the ∞ or h outcome of γ or γ puts x_α into M_γ as x_α is not reset and thus is not being restrained by another strategy. γ then puts x_α into B_δ at step (3)(a) or B if $x_\alpha \notin R_\alpha$ at step (3)(b) of the Q-strategy. Thus it follows that x_α enters B.

By inspection of strategies, if x is put in B_α by a lower priority node, x must come from a lower priority node γ working on a D-strategy as the Q- and S-strategies do not pick elements and put them into B_η for any η. The only elements D-strategies γ put into B_η for any η are the witnesses x_γ they chose. □

Lemma 2.4.4. *For every node α, if $x \in P_\beta$ is unused and is not reset as a witness, x does not go into B while α is restraining x.*

Proof. This follows by inspection of strategies and induction. Let x be an arbitrary unused element in P_β and let s be the stage that we choose x to be kept out of B. By our convention of unused/used, x is used after stage s.

Let γ be another node on the tree and let δ be its predecessor.

If γ is working on a D-strategy, it only puts x_γ or a number put into B_γ by its successor into B_δ. γ cannot pick x as x_γ before stage s as x would then be a used witness at stage s. Thus, $x_\gamma \neq x$. γ cannot pick x as x_γ at some stage $t > s$ as x_γ must be unused when chosen. The same reasoning shows that x cannot be put into B_γ by its successor as such a number comes from a lower priority D-node by the previous lemma.

If γ is working on a S-strategy, it only puts elements into B due to the ($\sharp$) requirement. If $x_\alpha \in V$, γ would either reset x_α by step (2) or (3) in the S strategy, set $x_\gamma = x_\alpha$ or have already defined $x_\gamma \neq x_\alpha$.

If γ is working on a Q-strategy, it either puts elements into B at step (4) or puts elements from M_γ into B_δ or B at step (3). At step (4), γ only puts unused elements into B to make B recursive on the complement of R_γ. If an arbitrary y goes into M_γ, it must enter B_γ first. By the previous lemma, y must be a x_η for a lower priority D-node η. η cannot pick x as x_η before stage s as x would be a used witness at stage s. Thus $x_\eta \neq x$ by assumption. η cannot pick x at some stage $t > s$ as x_η must be unused when chosen. □

Lemma 2.4.5. *Let α be a node on the true path and let s be the least stage such that the true path does not go to the left of α after stage s. P_α is either infinite at stage s or for infinitely many stages $t > s$, there are be unused elements added to P_α at stage t.*

Proof. By induction and inspection of strategies. If α is working on a D-requirement or S-requirement, $P_\alpha = P_\beta \setminus x_\alpha$ or $= P_\beta$.

If α is working on a Q-requirement and its outcome is c, $P_\alpha = R_\alpha$. α adds elements to R_α infinitely often at step (1) and does not reset R_α after some stage s. If the outcome for α is ∞, α adds an element to P_α^∞ infinitely often at step (3)(b). Since $P_\alpha = P_\alpha^\infty$ for the ∞ outcome, we are done. □

Lemma 2.4.6. *Each D_i requirement is satisfied.*

Proof. Let α be a node on the true path that is working on the D_i requirement and by Lemma 3.2, let s be the least stage such that x_α is not reset. If $\Delta_i(x_\alpha)$ does not converge or does not equal 0, the x_α does not enter B by Lemma 3.4 and the requirement is satisfied.

Now assume that $\Delta_i(x_\alpha)$ converges and is equal to 0. In the construction, the α-strategy puts x_α into B_γ where γ is its predecessor. As x_α does not reset after stage s, x_α is put into B by Lemma 3.3 and the requirement is satisfied. □

Lemma 2.4.7. *Either every S_V requirement is satisfied (and X is nonspeedable) or Y is nonspeedable.*

Proof. Suppose that c appears on the true path. Let α be the node on the true path whose outcome is c and let Q_h be the requirement that α is working on. As α's outcome is c, cofinitely many x's in M_α enter B before $h(x, \Phi_{M_\alpha}(x)) + 1$. Thus, higher priority requirements S_{V_i} must grab cofinitely many elements of M_α and put them into B before $h(x, \Phi_{M_\alpha}(x)) + 1$ due to (♯). In this case, we have the following equation: $B \cap R_\alpha = M_\alpha \cap R_\alpha = (V_0 \cup \cdots \cup V_i) \cap R_\alpha = X \cap R_\alpha$ (modulo finitely many elements). The last equality comes from (1) in the strategy for S_V (i.e. we commit to (♯) only if there is no x in both V and Y). By inspection of the Q_h strategy, we see that B is made recursive outside of R_α by step (4). Y is a split of this recursive part of B and thus it is recursive. Therefore Y is nonspeedable.

Suppose that c does not appear on the true path. By construction of the tree, for every S_V requirement, there is a node on the true path working it. By inspection of the strategy for S_V, either one of (1) - (4) holds or it commits to (♯) for cofinitely many stages in the construction. If it commits to (♯), we have that whenever x enters V, x enters B immediately. Therefore $g(x, s) = g(x, \Phi_V(x)) = \Phi_X(x)$ by the definition of g and thus S_V is satisfied.

If (1) holds, S_V is satisfied as $V \cap Y \neq \emptyset$ and $X \cap Y = \emptyset$. Thus $V \not\subseteq X$.

If (2) or (3) holds, a number in V is permanently kept out of B by the proof of Lemma 3.4. Thus $V \not\subseteq B$ and so $V \not\subseteq X$ and the requirement is satisfied.

If (4) holds, a number in V is permanently kept out of B by Lemma 3.4. Thus $V \not\subseteq B$ and so $V \not\subseteq X$ and the requirement is satisfied. □

Lemma 2.4.8. *Each Q_h requirement is satisfied.*

Proof. Let α be the first node on the true path f that is working on Q_h.

If c does not appear on f, each node α that is working on a Q_h strategy has outcome ∞ (or h) and thus there are infinitely many x's that are kept out of B until stage $h(x, \Phi_{M_\alpha}(x)) + 1$ or h is partial. Therefore, each Q_h is satisfied.

Suppose that c appears on f. If α appears before the c outcome and its successor is not c, we know its outcome is ∞ or h and thus its requirement is satisfied. If α's outcome is c, the next node γ on the true path works on the same requirement Q_h. Now suppose x enters M_γ at stage s. Such an x must exists. If no x ever enters M_γ from some point on, B would be recursive contradicting Lemma 3.6. Let $\eta \leq_L \alpha$ be working on a S-requirement. η cannot lie below the c outcome by construction of the priority tree. If η lies to the left or above the c outcome, η either restrains x permanently or has already restrained another element out permanently by (3)(b) in the strategy for S_V. By the proof of Lemma 3.2, there are only finitely many such active η's that restrain numbers from M_γ and each η only restrains one number. If $x \in M_\gamma$ eventually enters B, it enters B at stage $\geq$ the stage it enters B_δ (where δ is γ's predecessor) by Lemma 3.3. Thus $\Phi_B(x) > h(x, \Phi_{M_\gamma}(x))$ for infinitely many x's. By Lemma 3.2, $M_\gamma \subseteq^* B$ so Q_h is satisfied. If α appears after γ, the same reasoning as the previous case shows that $\Phi_B(x) > h(x, \Phi_{M_\alpha}(x))$ for infinitely many x's and $M_\alpha \subseteq^* B$ by Lemma 3.2. □

This ends the verification for the one split case.

In the one split case, we are either successful at making X nonspeedable by satisfying S-requirements or successful at concluding that Y is nonspeedable from the failure of a Q_h at satisfying its requirement using M_α. When working with more than one split, we would like to conclude that one of Y_0, Y_1,... is nonspeedable in the latter case. However, there could be V_i's for different pairs of splits (e.g. we could have various V_i's for X_0 and various V_j's for X_1) so the equation in R_α could involve more than one X (e.g. X_0, X_1, X_2).

To overcome this difficulty, we switch which side of the split we are making nonspeedable (e.g. from X_i to Y_i). Let Z_i be the side of the split we are making nonspeedable. In the case that α fails to satisfy its requirement, switching Z_i will give us some progress on higher priority requirements. In the proof of the one split case, having the equation: $B \cap R_\alpha = M_\alpha \cap R_\alpha = (V_0 \cup \cdots \cup V_i) \cap R_\alpha = X \cap R_\alpha$ (modulo finitely many elements) gave us progress on higher priority requirements by allowing us to conclude that Y is nonspeedable. As equations of such form are important for showing that we have made progress on higher priority requirements for the two split and general case, we give the following definition:

Definition 2.4.9. By *equation in* R_α, $B = M_\eta = V_{i_0} \cup ... \cup V_{i_k} = Z_{l_0} \cup ... \cup Z_{l_n}$ (where α, η are arbitrary nodes and i_0,...,i_k and $l_0, ..., l_n$ are arbitrary numbers), we mean that the following holds: $B \cap R_\alpha = M_\eta \cap R_\alpha = (V_{i_0} \cup ... \cup V_{i_k}) \cap R_\alpha = (Z_{l_0} \cup ... \cup Z_{l_n}) \cap R_\alpha$.

If the context is clear, we may drop "in R_α".

□

2.5 Two splits

In this section, we work with the case where we are recursively given two splits: X_0, Y_0 and X_1, Y_1. We refer to Z_i as the half of the split that we are making nonspeedable.

Recall that in the one split case, we are either successful at making X nonspeedable by satisfying S-requirements or successful at concluding that Y is nonspeedable from the failure of an Q_h at satisfying its requirement using M_α and the equation in R_α (see Definition 3.9) that its failure implies. However when working with two splits, there could be V_i's for different pairs of splits (e.g. we could have various V_i's for X_0 and various V_j's for X_1) so the equation in R_α could involve X_0 and X_1.

To overcome this obstruction, we use the following idea: If M_α fails to satisfy its requirement (and we have an equation in R_α as before e.g. $X_0 \cup X_1 = B$), we switch Z_1 from X_1 to Y_1. If another M_γ fails to satisfy its requirement where $\gamma \supseteq \alpha$, we have an equation in R_γ, e.g. $X_0 \cup Y_1 = B$. We also switch Z_0 from X_0 to Y_0 and reset Z_1 back to X_1. From these equations together and since $R_\gamma \subseteq R_\alpha$, we can conclude the following equation in R_γ: $X_0 = B$. If we continue to see more nodes $\nu \supseteq \gamma$ fail to satisfy their requirements using M_ν, our method of switching the value of Z_i lets us conclude $Y_0 = B$ in some $R_\eta \subseteq R_\gamma$ (for some node η). This will allow us to prove that only a fixed number of switchings can occur. These switchings will be explained in detail in the proof.

Lemma 2.5.1. *For any two recursive functions f, g, there is a speedable set B such that if X_0, Y_0 is the split of B given by f and X_1, Y_1 is a split of B given by g then for each i, at least one of X_i or Y_i is nonspeedable.*

Proof. The proof is by a tree construction like in the one split case.

Requirements

To make B speedable, we have similar requirements Q_h as in the one split case:

$$Q_h : (\exists M \subseteq^* B)(\exists^\infty x)(\Phi_B(x) > h(x, \Phi_M(x)) \vee \text{ switch some } Z_i$$

Switching some Z_i is a form of progress on higher priority strategies just as concluding that Y is nonspeedable was a form of progress for the one split case.

To make X_i or Y_i nonspeedable, we have the following requirements:

$$S_i : (\exists x)(x \in V \wedge x \notin Z_i) \text{ or } (\forall x)[x \in Z_i \Rightarrow \Phi_{Z_i}(x) \leq g_i(x, \Phi_V(x))]$$

and its subrequirements, where g_i is as defined below:

$$S_{i,V} : (\exists x)(x \in V \wedge x \notin Z_i) \text{ or } (\forall x)[x \in Z_i \Rightarrow \Phi_{Z_i}(x) \leq g_i(x, \Phi_V(x))]$$

Let $g_i(x, s)$ be the least t that x enters X_i or Y_i if x is in B_s. Otherwise, we define $g_i(x, s)$ to be 0. Observe that g_i is total and recursive. We also require S_i to be of higher priority than S_j if $i < j$. We occasionally drop the i subscript if the context is clear.

We refer to S_i as the parent requirement of $S_{i,V}$.

Again, we have the D_i requirements as before to help the Q-requirements.

Strategies

The strategies for Q-, S- and D-requirements are the same as before except for the following differences:

1. Each α node has a current Z_i^α that it is working on. Each Z_i^α is initially defined to be X_i but may switch to Y_i in the course of the construction.

2. The strategy for $S_{i,V}$ is the same as the strategy for S_V in the one split case except that instead of X and Y, we have Z_i^α and the other half of the i^{th} pair.

Outcomes

Each node α working on a D-strategy has outcomes: a (for acts) and d (for diverge). On the a outcome, $\Delta_i(x)$ converges and equals to 0 and α would like to put x into B. Along the d outcome, either $\Delta_i(x)$ diverges or is not equal to 0.

Each node α working on a Q-strategy has outcomes: ∞, c and h. Along the ∞ outcome, infinitely many x's in M_α enter B after stage $h(x, \Phi_{M_\alpha}(x)) + 1$. Along the c outcome, cofinitely many x's in M_α enter B before $h(x, \Phi_{M_\alpha}(x)) + 1$ and we switch Z_i. Along the h outcome, we see that h is partial.

Each node α working on a S_i-strategy has a blank outcome. In the general case, it is important to keep track of the overall timing of when x's from V's enter B due to actions of $S_{i,V}$ strategies. Here, S_i keeps track of the state of its child nodes.

Each node α working on a $S_{i,V}$-strategy has outcomes: k and s. Along the k outcome, either there is some x in both V and Y or α can keep some element in V out of B. Along the s outcome, α cannot achieve $(\exists x)(x \in V \wedge x \notin X)$ and we try to achieve $x \in X \Rightarrow \Phi_X(x) \leq g(x, \Phi_V(x))$ for all x.

Switchings of Z_0^α and Z_1^α

The new ingredient in the two split case that goes beyond the one split case is our way of assigning Z_i^α to each node α so that if we get an equation in R_α from the c outcome, the equation in R_α gives us progress on higher priority strategies. In the one split case, we proved that we could eventually get Y to be recursive. Here, the situation is more complicated and we switch the Z_0 and Z_1 so that we can obtain a contradiction from getting too many c outcomes for different nodes on the same path.

The main idea is the following: A c outcome for a Q-node causes a switch of the highest indexed Z_i that is equal to X_i and resets all higher indexed (for $k > i$) sets Z_k to X_k. If such a Z_i does not exist, we do not switch anything. We will prove in the verification that such a Z_i will always exist for c outcomes on the true path (see Lemma 4.3). Switching at some node α effects Z_i^γ for $\gamma \supseteq \alpha$ unless a switching occurs at a later node $\nu \supseteq \alpha$.

Configuration	Node	Value of Z_0	Value of Z_1	Equation in R_{node}
0	before β	X_0	X_1	no equation by assumption
1	β	X_0	Y_1	$X_0 \cup X_1 = B$
2	α	Y_0	X_1	$X_0 \cup Y_1 = B$
3	δ	Y_0	Y_1	$Y_0 \cup X_1 = B$
4	γ	Y_0	Y_1	$Y_0 \cup Y_1 = B$

Figure 2.2: A table summarizing the possible switching that can occur for the two split case. The "Value of Z_0 (or Z_1)" for configuration n refers to the value of Z_0^ν (Z_1^ν) for nodes ν between the node for configuration n and the node for configuration $n+1$. An equation in R_{node} for configuration n refers to the equation in R_η where η is the value of the node for configuration n.

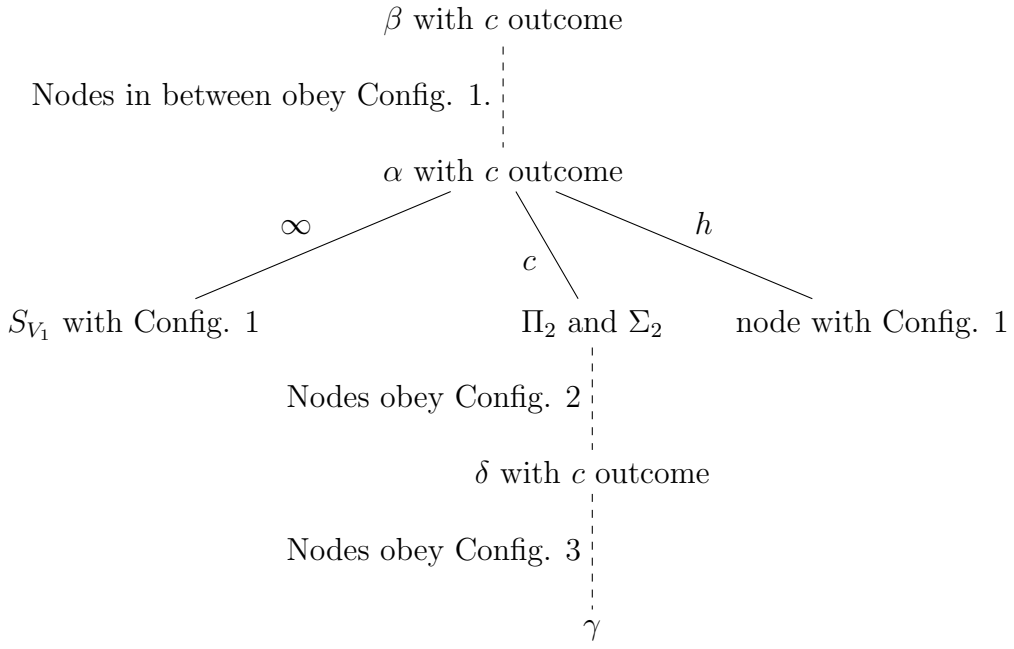

Figure 2.3: In this figure, we assume that there is no node with true outcome c above γ except for β, α and δ. Assuming that this figure signifies a portion of the true path, the dashed edges between nodes represents the true path in between the two nodes.

Now, we assume that $\beta \subseteq \alpha \subseteq \delta \subseteq \gamma$ are on the true path with true outcome c (as in Figure 3 above). We also assume that there is no other Q_h requirement node with true outcome c between them and β is the first node on the true path working on a Q_h-requirement with true outcome c and so Z_0^β and Z_1^β are equal to their initial values, X_0 and X_1 respectively. In other words, we assume that $\beta, \alpha, \delta, \gamma$ are the first, second, third and fourth node (respectively) on the true path with true outcome c. The possibilities are the following (summarized in Figure 2 above):

1. As c is the true outcome of β, we have the following equation in R_β as in the one split case: $X_0 \cup X_1 = B$. We switch Z_1 from X_1 to Y_1.

2. As c is the true outcome of α, we have the following equation in R_α as in the one split case: $X_0 \cup Y_1 = B$. We switch Z_0 from X_0 to Y_0 and reset Z_1, i.e. switch Z_1 from Y_1 back to X_1.

3. As c is the true outcome of δ, we have the following equation in R_δ as in the one split case: $Y_0 \cup X_1 = B$. We switch Z_1 from X_1 to Y_1.

4. (cannot occur on the true path) As c is the true outcome of γ, we have the following equation in R_γ as in the one split case: $Y_0 \cup Y_1 = B$. We do not switch anything.

Possibility (4) cannot occur due to the following reasoning: Suppose for a contradiction that possibility (4) occurs. As $\beta \subseteq \alpha \subseteq \delta \subseteq \gamma$, we have $R_\beta \supseteq R_\alpha \supseteq R_\delta \supseteq R_\gamma$. Therefore, we have the following equations in R_γ: $X_0 \cup X_1 = B$, $X_0 \cup Y_1 = B$, $Y_0 \cup X_1 = B$ and $Y_0 \cup Y_1 = B$. The first two equations implies the following equation in R_γ: $X_0 = B$. The last two equations implies the following equation in R_γ: $Y_0 = B$. The satisfaction of some D_i requirement on a node after γ puts some element x into $B \cap R_\gamma$ as requirements being satisfied by nodes on the true path after $\gamma^\frown\langle c\rangle$ only take witnesses from R_γ. However, this would mean that x enters both X_0 and Y_0 by the two equations in R_γ: $X_0 = B$ and $Y_0 = B$. This is a contradiction because X_0 and Y_0 are disjoint as X_0 and Y_0 form a split of B.

Priority Tree

Fix a recursive ordering of the D-, Q- and S-requirements, respectively, where S_i is of higher priority of than $S_{i,V}$.

Let $\Lambda = \{\infty, c, h, k, s, a, d\}$ with ordering $\infty < c < h < k < s < a < d$. The tree is a subset of $\Lambda^{<\omega}$.

We define by recursion a function G such that G assigns to each α the list of Z_0^α and Z_1^α it is working on in the construction (e.g. $G(\alpha) = (X_0, Y_1)$). We also keep track of two lists $L_1(\alpha)$ and $L_2(\alpha)$ for each node. L_1 keeps track of the S-requirements that have appeared so far (but gets reset at every appearance of a c outcome) and L_2 keeps track of S-requirements that need to be repeated and we remove a requirement from this list once it has been repeated.

We assign requirements to nodes as well as define G, L_1, L_2 by recursion at the same time. For the empty node, assign the highest priority Q_h requirement to α. Let $G(\emptyset) = (X_0, X_1)$ and let $L_1(\emptyset) = L_2(\emptyset) = \emptyset$. Suppose that we have assigned a requirement to $\beta = \alpha \upharpoonright (|\alpha|-1)$, which we will call the β requirement and suppose that we have defined $G(\beta), L_1(\beta)$ and $L_2(\beta)$. We now assign a requirement to α and define $G(\alpha), L_1(\alpha)$ and $L_2(\alpha)$.

Defining $G(\alpha)$

Ask whether β is a node working on some Q_h with successor c. If not, let $G(\alpha)$ be $G(\beta)$. If so, let k be 1 if X_1 appears in $G(\beta)$ and let k be 0 otherwise. We have the following possibilities:

1. If $k = 0$ and Y_0 appears in $G(\beta)$, define $G(\alpha)$ to be $G(\beta)$.
2. If $k = 0$ and X_0 appears in $G(\beta)$, define $G(\alpha)$ to be (Y_0, X_1).
3. If $k = 1$, define $G(\alpha)$ to be $G(\beta)$ with X_1 switched to Y_1, i.e. if $G(\beta) = (Z', X_1)$ then define $G(\alpha)$ to be (Z', Y_1).

Defining $L_1(\alpha)$ and $L_2(\alpha)$

Ask whether β is a node working on some Q_h with successor c. If so, let $L_2(\alpha) = L_1(\beta)$ and set $L_1(\beta) = \emptyset$. Otherwise, ask whether β is a node working on some S-requirement. If so, let $L_1(\alpha)$ be $L_1(\beta)$ with β's requirement affixed at the end. If $L_2(\beta)$ is not empty and this S-requirement is in $L_2(\beta)$, let $L_2(\alpha)$ be $L_2(\beta)$ with this S-requirement removed. Otherwise, let $L_1(\alpha) = L_1(\beta)$ and let $L_2(\alpha) = L_2(\beta)$.

Assigning a requirement to α

If the β-requirement is an Q-requirement with outcome c, assign the β-requirement to α and let its successors be ∞, c and h.

Otherwise, check if $L_2(\alpha)$ is empty or not. If $L_2(\alpha)$ is not empty, assign the highest priority S requirement in $L_2(\alpha)$ to α. If this requirement is a S_i-requirement, let its successor be the blank outcome and if this requirement is a S_V-requirement, let its successors be k and s.

If $L_2(\alpha)$ is empty, we assign a requirement to α based on the type of requirement assigned to β-requirement: If the β-requirement is a Q-requirement, assign the highest priority D-requirement that has not been assigned so far and let its successors be a and d. If the β-requirement is a D-requirement, assign the highest priority S-requirement that has not been assigned so far and let its successors be k and s if it is a S_V requirement and let its successor be the blank outcome if it is a S_i requirement. Otherwise, assign the highest priority Q-requirement that has not been assigned so far and let its successors be ∞, c and h.

Construction

We call a node working on an S-strategy *active* at stage s if the node has acted at some stage $t < s$ and has not been cancelled (or reset).

At stage s, define δ_s (an approximation to the true path) by recursion as follows. Suppose that $\delta_s \upharpoonright e$ has been defined for $e < s$. Let α be the last node of $\delta_s \upharpoonright e$. We now define $\delta_s \upharpoonright (e+1) \supseteq \delta_s \upharpoonright e$.

1. If α is a Q-node, look to see if it is waiting for a number x to go into B_β, i.e. there is some x in M_α that has not entered B_β. If so, look to see if an active node η to to the left or below $\alpha^\frown\langle\infty\rangle$ would like to put x in (or keep x out). Let η act. If there are no remaining numbers that α is waiting on, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle c\rangle$. Otherwise, look to see if there is some z in the remaining numbers such that $h(z, \Phi_{M_\alpha}(z))$ has converged and our current stage is greater than $h(z, \Phi_{M_\alpha}(z)) + 1$. If so, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle\infty\rangle$. If not and some x was pulled, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle c\rangle$. Otherwise, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle h\rangle$.

2. If α is a S_V-node, look to see if x_α has been defined. If so, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle k\rangle$. Otherwise, go through steps (1) - (4) in the strategy for S_V. If one of (1) - (4) holds and we can define x_α, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle k\rangle$. Otherwise, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle s\rangle$.

3. If α is a D-node, look to see whether $\Delta_i(x_\alpha)$ has converged and equals 0. If so, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle a\rangle$. Otherwise, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle d\rangle$.

Reset nodes to the right of δ_s and let active S-nodes to the left of δ_s or below the ∞ outcome for one of the nodes in δ_s act. At substage $t \leq s$, let $\delta_s \upharpoonright t$ act according to its description in the strategies above, i.e. $\delta_s \upharpoonright t$ enumerates numbers in its sets and extends their definitions.

Verification

Recall that the true path denotes the leftmost path traveled through infinitely often by the construction.

Lemma 2.5.2. *For every node α, if x is an unused witness in P_β and is not reset as a witness, x does not go into B while α is restraining x. For every node α, if $x_\alpha = x_\gamma$ for another node γ, we can successfully keep x_α out of B. If α is on the true path, $M_\alpha \subseteq^* B$. If α is on the true path, we only reset its witness finitely many times.*

Proof. The proof follows the same reasoning as in the one split case. □

Lemma 2.5.3. *There cannot be more than three nodes with true outcome c on the true path.*

Proof. Suppose otherwise and let $\beta \subseteq \alpha \subseteq \delta \subseteq \gamma$ be the first four nodes on the true path with outcome c. By our scheme of switching between X and Y, we obtain the following equations: $X_0 \cup X_1 = B$ (in R_β), $X_0 \cup Y_1 = B$ (in R_α), $Y_0 \cup X_1 = B$ (in R_δ) and $Y_0 \cup Y_1 = B$ (in R_γ). By our choice of β, α, δ and γ, we have $R_\beta \supseteq R_\alpha \supseteq R_\delta \supseteq R_\gamma$ and thus all of these equations are true in R_γ. The first two equations give the following equation in R_γ: $X_0 = B$ and the last two equations give the following equation in R_γ: $Y_0 = B$. At some stage, the satisfaction of some D_i requirement puts some element x into $B \cap R_\gamma$ as requirements being satisfied by nodes on the true path after the c outcome of γ only take witnesses from R_γ. However, this would mean that x enters both X_0 (from the equation in R_γ: $X_0 = B$) and Y_0 (from the equation in R_γ: $Y_0 = B$). As X_0 and Y_0 are disjoint, we have obtained our contradiction. □

Lemma 2.5.4. *We only switch Z_0 and Z_1 finitely many times on the true path.*

Proof. This follows immediately from the previous lemma as the construction does not switch the value of Z_0 or Z_1 unless it meets a c outcome. □

Lemma 2.5.5. *On the true path, for every requirement, there is a node that works on it.*

Proof. It suffices to show that there exists an α on the true path such that for all $\nu \supseteq \alpha$ on the true path, $L_2(\alpha)$ is empty. Let γ be an arbitrary node and let δ be the successor of γ. By construction of L_2, either L_2 is a finite set or is empty. $L_2(\gamma)$ is not empty if and only if $L_2(\delta)$ is not empty or δ is a node working on a Q_h requirement with outcome c. The latter case can only occur less than three times by Lemma 4.3. The former case only occurs finitely many times as the cardinality of the value of L_2 strictly decreases as we go down a path unless we met another node working on a Q_h requirement with outcome c. However, there are only finitely many such nodes on the true path so the lemma follows by setting α to be the first node such that $L_2(\alpha)$ is empty and no nodes after α has the c outcome on the true path. □

Lemma 2.5.6. *For all i, the D_i requirement is satisfied.*

Proof. This follows by Lemma 4.2 and similar reasoning as in the one split case. □

Lemma 2.5.7. *S_0 and S_1 are both satisfied.*

Proof. If we never switch the value of Z_0 to Y_0 on the true path, we never reset the $S_{0,V}$ strategies and as there is a node working on $S_{0,V}$ for every V on the true path, the S_0 requirement is satisfied. If we do switch, we finish switching at some stage s by Lemma 4.4. We do not reset the value of Z_0 for S_0 strategies again after we finish switching so there is a node working on $S_{0,V}$ with the final value of Z_0 for every V on the true path and thus the S_0 requirement is satisfied by Lemma 4.5 and by our scheme of repeating S_V-requirements using list L_2.

The same reasoning works for S_1 as well. By Lemma 4.4, Z_1 finishes switching at some stage t. We do not reset the value of Z_1 for S_1 strategies again after we finish switching so

there is a node working on $S_{1,V}$ with the final value of Z_1 for every V on the true path and thus the S_1 requirement is satisfied. □

Lemma 2.5.8. *For all h, Q_h is satisfied.*

Proof. This follows from Lemma 4.3. As there can only be three nodes on the true path with outcome c, some α on the true path working on Q_h must get the ∞ outcome, i.e. there are infinitely many elements in M_α that are kept out of B until stage $h(x, \Phi_{M_\alpha}(x)) + 1$. □

□

2.6 The general case

In this section, we prove Theorem 2.3.

We would like to use the same line of argument as in the two split case where having too many equations (as in the sense of Definition 3.9) leads to a contradiction. Here, we are dealing with infinitely many splits. During the construction, various sets will be proven to be recursive and various switchings will occur. The worry is that this may fill up the whole universe and then there would be no way to make B speedable or even nonrecursive. Lemma 5.5 is devoted to establishing that such a situation does not occur.

Proof of Theorem 2.3. We deal with all possible splits of B. Recursively order all pairs X_i, Y_i of disjoint r.e. subsets of B. As before, we let Z_i be the split we are currently trying to make nonspeedable. To start, Z_i is X_i. For each node α, the priority tree assigns which Z_i α is working on.

Requirements

We have Q_h requirements like in the one split case:

$$Q_h : (\exists M \subseteq^* B)(\exists^\infty x)(\Phi_B(x) > h(x, \Phi_M(x)) \vee \text{ switch some } Z_i$$

Switching some Z_i is a form of progress on higher priority strategies just as concluding that Y is nonspeedable was a form of progress for the one split case.

We have S-requirements to make Z_i nonspeedable:

$$S_i : (\exists g_i)(\forall r.e.\ V) \text{ either } V \not\subseteq Z_i \text{ or } (\forall x)[x \in Z_i \Rightarrow \Phi_{Z_i}(x) \leq g_i(x, \Phi_V(x))]$$

and their subrequirements, $S_{i,V}$:

$$S_{i,V} : (\exists x \in V)x \notin Z_i \text{ or } (\forall x)(x \in Z_i \Rightarrow \Phi_{Z_i}(x) \leq g_i(x, \Phi_V(x)))$$

We refer to S_i as the parent requirement of $S_{i,V}$.

We define g_i at the parent node working on S_i. We also require S_i to be of higher priority than S_j iff $i < j$. We occasionally drop the i subscript if the context is clear.

Again, we have the D_i requirements as before to help the Q-requirements.

Strategies

Let $f_i(x,s)$ be the least y greater than every $h(x,t)$, where h belongs to a higher priority Q_h that does not have the h outcome and greater than every $g_j(x,t)$ where g_j belongs to a higher priority S_j for all $t \leq s$. Let $g_i(x,s)$ be the least y greater than $f_i(x,s)$ and also greater than the least stage t that x enters either X_i or Y_i if x has entered $B_{f_i(x,s)}$. This definition of g_i is needed in the verification to show that if α is a Q-node, child nodes for lower priority S_i requirements cannot injure α (see first paragraph of the proof of Lemma 5.3).

The strategies for Q_h, $S_{i,V}$ and D_i are the same as in the one and two split cases except that the $S_{i,V}$ strategy has the following differences:

1. Each node α has its version of the Z_i^α, f_i^α, g_i^α, determined by the priority tree. Instead of X and Y in the strategy for S_V in the one split case, we work on Z_i^α and the other side of the split for $S_{i,V}$.

2. The ($\sharp$) commitment we use for the general case in the $S_{i,V}$ requirement is:

$$\text{When } x \text{ enters } V \text{ at stage } s, \text{ we put } x \text{ into } B \text{ at stage } f_i(x,s). \qquad (\sharp)$$

Switching of the Z_i^α and reduction to the n-split case

At an Q-node α, we will prove in the verification that any equation obtained in R_α will only involve Z_i's where i is such that the S_i requirement is assigned to a node above α. We will only introduce new splits to be considered *only* after α's requirement has been satisfied by an ∞ outcome (i.e. until some node assigned to α's requirement gets the ∞ outcome, we do not assign any new S_i-requirements to any node on this path).

Let α be an arbitrary node on the tree working on some Q_h-requirement and let $Z_0, ..., Z_n$ be a listing of Z_i's such that the S_i requirement is assigned to some node above α.

The switching of the Z_i^α is similar to the two split case. A c outcome for a Q-node causes a switch of the highest indexed Z_i that is equal to X_i and rests all higher indexed (for $k > i$) sets Z_k and X_k. The reasoning behind this switching is to systematically switch so that if we just look at $Z_0, ..., Z_l$ and all of the switches have occurred concerning $Z_0, ..., Z_l$, we obtain equations $Z_0 \cup ... \cup Z_l = B$ for every combination of values for $Z_0, ..., Z_l$. This is necessary for proving that we can make B speedable.

A typical situation is as in Figure 4 below. There are nodes working on the same Q_h requirement and as in the two split case, the Q_h requirement is continually being assigned to nodes along this path until one of the nodes γ working on this Q_h requirement gets an ∞ outcome. Until γ appears, all nodes appearing after α on this path are working on a same fixed number of splits as α. If γ never appears, the situation for nodes appearing after α is as in the n-split case. Like in the 2 split case, we can argue as in Lemma 4.3 that we obtain a contradiction from having too many equations resulting from too many nodes working on the Q_h requirement with outcome c. Thus, γ has to appear.

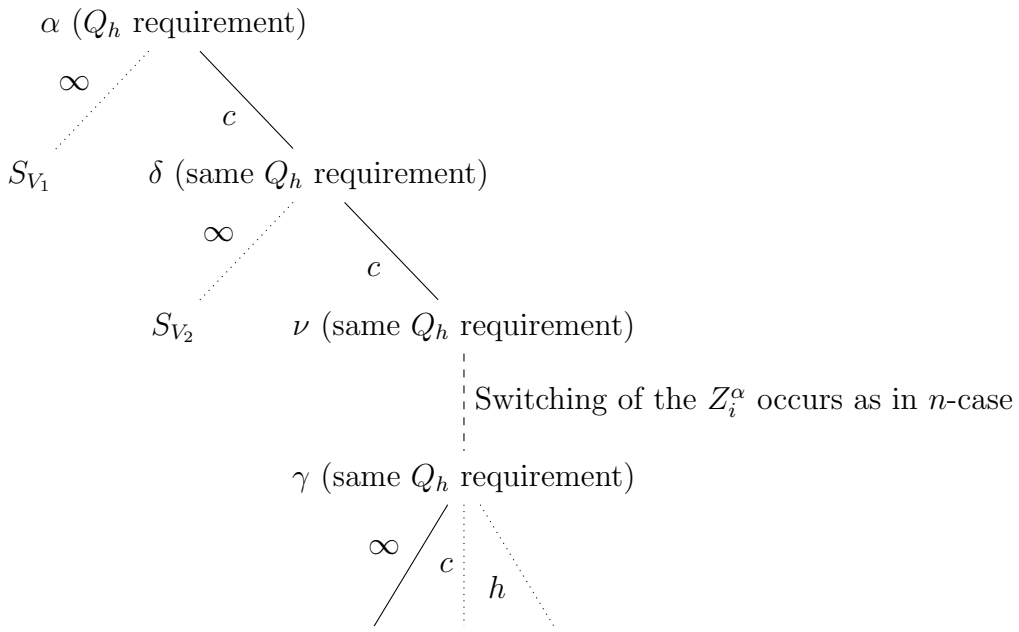

Figure 2.4: Assuming that this figure signifies a portion of the true path, a dotted line signifies that the true path does not go this way and the dashed edges between nodes signifies the true path in between the two nodes. Here, we assume that all Q-nodes on the true path between ν and γ that work on the same Q_h requirement as α has true outcome c. Technically, there are other nodes working on S-requirements between α and δ and δ and ν but we have not included them in the picture.

Outcomes

Each node α working on a D-strategy has outcomes: a (for acts) and d (for diverge). On the a outcome, $\Delta_i(x)$ converges and equals to 0 and α would like to put x into B. Along the d outcome, either $\Delta_i(x)$ diverges or is not equal to 0.

Each node α working on a Q-strategy has outcomes: ∞, c and h. Along the ∞ outcome, infinitely many x's in M_α enter B after stage $h(x, \Phi_{M_\alpha}(x)) + 1$. Along the c outcome, cofinitely many x's in M_α enter B before $h(x, \Phi_{M_\alpha}(x)) + 1$ and either we have the means to conclude that Z_i is nonspeedable or we switch Z_i. Along the h outcome, we see that h is partial.

Each node α working on a S_i-strategy has outcomes: *split* or $finite$. Along the *split* outcome, we see that X_i and Y_i form a split of B and define f_i and g_i for the child-nodes of S_i to work on. Along the $finite$ outcome, we see that X_i and Y_i do not form a split of B. Under the $finite$ outcome, there are no child nodes of S_i.

Each node α working on a $S_{i,V}$-strategy has outcomes: k and s. Along the k outcome, either there is some x in both V and Y or α can keep some element in V out of B. Along the

s outcome, α cannot achieve $(\exists x)(x \in V \wedge x \notin X)$ and we try to achieve $x \in X \Rightarrow \Phi_X(x) \leq g(x, \Phi_V(x))$ for all x.

Priority Tree

Fix a recursive ordering of the D-, Q- and S-requirements where S_i is of higher priority of than $S_{i,V}$.

Let $\Lambda = \{\infty, c, h, split, finite, k, s, a, d\}$ with ordering $\infty < c < h < split < finite < k < s < a < d$. The tree is a subset of $\Lambda^{<\omega}$.

As in the two split case, we define by recursion a function G such that G assigns to each α the list of Z_i^α's it is working on in the construction (e.g. $G(\alpha) = (X_0, Y_1, X_2)$). As in the two split case, we also keep track of two lists $L_1(\alpha)$ and $L_2(\alpha)$ for each node. L_1 keeps track of the S-requirements that have appeared so far (but gets reset at every appearance of a c outcome) and L_2 keeps track of S-requirements that need to be repeated.

We assign requirements to nodes as well as define G, L_1, L_2 by recursion at the same time. For the empty node, assign the highest priority Q_h requirement to α. Let $G(\emptyset) = (X_0)$ and let $L_1(\emptyset) = L_2(\emptyset) = \emptyset$. Suppose that we have assigned a requirement to $\beta = \alpha \upharpoonright (|\alpha| - 1)$, which we will call the β requirement and defined $G(\beta), L_1(\beta)$ and $L_2(\beta)$. We now assign a requirement to α and define $G(\alpha), L_1(\alpha)$ and $L_2(\alpha)$.

Defining $G(\alpha)$

Ask whether β is a node working on some Q_h with successor c. If so, let k be the largest index such that X_k appears in $G(\beta)$ and let $\overrightarrow{Z_0}$ and $\overrightarrow{Z_1}$ be such that $G(\beta) = \overrightarrow{Z_0} X_k \overrightarrow{Z_1}$. Note that we will prove that k always exists if α is on the true path. If k does not exist, define $G(\alpha)$ to be $G(\beta)$. If k exists, define $G(\alpha)$ to be $G(\beta)$ with X_k switched to Y_k and X_i's and Y_i's in $\overrightarrow{Z_1}$ reset to be X_i i.e. $G(\alpha) = \overrightarrow{Z_0} Y_k \overrightarrow{X_1}$ where $\overrightarrow{X_1}$ is the X-side of the splits in $\overrightarrow{Z_1}$. If β is not working on some Q_h with successor c, ask whether β is a node working on some S_i. If so and both X_i and Y_i do not appear in $G(\beta)$, let $G(\alpha)$ be $G(\beta)$ with X_i appended to the end of $G(\beta)$'s list. If not, define $G(\alpha)$ to be $G(\beta)$.

Defining $L_1(\alpha)$ and $L_2(\alpha)$

Ask whether β is a node working on some Q_h with successor c. If so, let $L_2(\alpha) = L_1(\beta)$ and set $L_1(\beta) = \emptyset$. Otherwise, ask whether β is a node working on some S-requirement. If so, let $L_1(\alpha)$ be $L_1(\beta)$ with β's requirement affixed at the end. If $L_2(\alpha)$ is not empty and this S-requirement is in $L_2(\alpha)$, let $L_2(\alpha)$ be $L_2(\beta)$ with this S requirement removed. Otherwise, let $L_1(\alpha) = L_1(\beta)$ and let $L_2(\alpha) = L_2(\beta)$.

Assigning a requirement to α

If the β-requirement is a Q-requirement with outcome c, assign the β-requirement to α and let its successors be ∞, c and h.

Otherwise, check if $L_2(\alpha)$ is empty or not. If $L_2(\alpha)$ is not empty, assign the highest priority S requirement in $L_2(\alpha)$ to α. If this requirement is an S_i-requirement, let its successors be the *split* and $finite$ outcomes and if this requirement is a S_V-requirement, let its successors be k and s.

If $L_2(\alpha)$ is empty, we assign a requirement to α based on the type of requirement assigned to β-requirement: If the β-requirement is a Q-requirement, assign the highest priority D-requirement that has not been assigned so far and let its successors be a and d. If the β-requirement is a D-requirement, assign the highest priority S-requirement that has not been assigned so far and such that β does not extend the $finite$ outcome for a node assigned to the parent requirement of this S-requirement (if this S-requirement is not the parent requirement itself) and let its successors be k and s if it is an S_V requirement and let its successors be *split* and $finite$ if it is an S_i requirement. Otherwise, assign the highest priority Q-requirement that has not been assigned so far and let its successors be ∞, c and h.

Construction

We call a node working on an S-strategy *active* at stage s if the node has acted at some stage $t < s$ and has not been cancelled (or reset).

At stage s, define δ_s (an approximation to the true path) by recursion as follows. Suppose that $\delta_s \upharpoonright e$ has been defined for $e < s$. Let α be the last node of $\delta_s \upharpoonright e$. We now define $\delta_s \upharpoonright (e+1) \supseteq \delta_s \upharpoonright e$.

1. If α is a Q-node, look to see if it is waiting for a number x to go into B_β, i.e. there is some x in M_α that has not entered B_β. If so, look to see if an active node η to to the left or below $\alpha^\frown\langle\infty\rangle$ would like to put x in (or keep x out). Let η act. If there are no remaining numbers that α is waiting on, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle c\rangle$. Otherwise, look to see if there is some z in the remaining numbers such that $h(z, \Phi_{M_\alpha}(z))$ has converged and our current stage is greater than $h(z, \Phi_{M_\alpha}(z)) + 1$. If so, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle\infty\rangle$. If not and some x was pulled, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle c\rangle$. Otherwise, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle h\rangle$.

2. If α is a S_V-node, look to see if x_α has been defined. If so, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle k\rangle$. Otherwise, go through steps (1) - (4) in the strategy for S_V. If one of (1) - (4) holds and we can define x_α, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle k\rangle$. Otherwise, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle s\rangle$.

3. If α is a S_i-node, look to see if X_i and Y_i seem to form a split of B. If so, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle split\rangle$. Otherwise, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle finite\rangle$.

4. If α is a D-node, look to see whether $\Delta_i(x_\alpha)$ has converged and equals 0. If so, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle a\rangle$. Otherwise, let $\delta_s \upharpoonright (e+1) = (\delta_s \upharpoonright e)^\frown\langle d\rangle$.

Reset nodes to the right of δ_s and let active S-nodes to the left of δ_s or below the ∞ outcome for one of the nodes in δ_s act. At substage $t \leq s$, let $\delta_s \upharpoonright t$ act according to its

description in the strategies above, i.e. $\delta_s \upharpoonright t$ enumerates numbers in its sets and extends their definitions.

Verification

Recall that the true path denotes the leftmost path travelled through infinitely often. In the following lemmas, we may drop "in R_α" when referring to an equation. In the proofs below, we only deal with a sequence of equations in R_α where α's are on the same path. As $R_\alpha \subseteq R_\beta$ if $\beta \subseteq \alpha$, these equations in different R_α's are true in their intersection (see Lemma 5.4). We also drop the superscript α when referring to Z_i^α as the version of Z_i we refer to will be clear from context. The first few lemmas lead to proving that there is a node working on every requirement on the true path.

Lemma 2.6.1. *For every node α, if x is an unused witness in P_β and is not reset as a witness, x does not go into B while α is restraining x. For every node α, if $x_\alpha = x_\gamma$ for another node γ, we can successfully keep x_α out of B. If α is on the true path, $M_\alpha \subseteq^* B$. If α is on the true path, we only reset its witness finitely many times.*

Proof. The proof follows the same reasoning as in the one split case. □

The next few lemmas use the following definition.

Definition 2.6.2. If α working on Q_h has outcome c, we are unable to achieve (†) in the Q_h requirement for infinitely many x's. Thus S_{i,V_j} strategies must put cofinitely many of these x's into B before stage $h(x, \Phi_{M_\alpha}(x)) + 1$ due to (♯). Let $S_{i_0,V_{j_0}}, \cdots, S_{i_k,V_{j_k}}$ be a listing of the S-strategies involved (i.e. the active S-nodes below the ∞ outcome of α). We only commit to (♯) when there is no x in both V and the other half of B, in which case $V_{j_0} \cup \cdots \cup V_{j_k} = Z_{i_0} \cup \cdots \cup Z_{i_k}$. Therefore, modulo finitely many elements we have the following equation in R_α: $B = M_\alpha = V_{j_0} \cup \cdots \cup V_{j_k} = Z_{i_0} \cup \cdots \cup Z_{i_k}$. We refer to an equation in R_α: $B = X_{i_0} \cup \cdots \cup X_{i_j} \cup Y_{k_0} \cup \cdots \cup Y_{k_l}$ as an *equation* obtained from the c outcome if we obtained this equality directly in the way described above. We refer to an equation in R_α: $B = X_{i_0} \cup \cdots \cup X_{i_j} \cup Y_{k_0} \cup \cdots \cup Y_{k_l}$ as an *reduced equation* if equations obtained from the c outcome imply it.

Remark 2.6.3. In the equations mentioned above (e.g. $V_{j_0} \cup \cdots \cup V_{j_k} = M_\alpha$), this is not completely correct as there could be weak $S_{i,V}$ requirements where S_i is stronger than α that put elements into M_α. Let ν be one of these weak requirements. ν could pretend to put x into one side of the split when x enters B but then switch to the other side. Once it sees an x that goes into B quickly, it puts x into the other side. Such x's make our equation false. However, we still have that inside M_α, $B = V_{j_0} \cup \cdots \cup V_{j_k}$ except for numbers that are put into B by such ν.

Now, we are fine with this because x's from D-requirements are not effected by these false η's. This is because nodes working on the D-requirements will wait until requirements to the left settle down so D can pick its x to avoid the numbers that these false η's are

involved with. D prohibits x from being used by the weaker S_V's and thus the main point is that the equations mentioned above hold for all x's from a D-requirement.

Lemma 2.6.4. *Let α be a node working on the Q_h requirement. If α's outcome is c, the equation obtained from the c outcome can only involve $S_{i,V}$ requirements where S_i is of higher priority than Q_h.*

Proof. If S_i lies below or to the right of α, the lemma follows by our construction of $g_i(x, s)$. We only need $x \in M_\alpha$ to stay out of B before $h(x, \Phi_{M_\alpha}(x)) + 1$ which is strictly less than $g_i(x, \Phi_V(x))$ for x's such that $\Phi_{M_\alpha}(x) \leq \Phi_V(x)$. This is why we need our definition of g_i.

Now let γ be a node on to the left of α working on a $S_{i,V}$ requirement with its parent node to the left of α. We now prove that $S_{i,V}$ cannot be involved in the equation we obtain from α's c outcome. To occur in the equation, $S_{i,V}$ must commit to ($\sharp$) and some $x \in M_\alpha$ is put in B before $h(x, \Phi_{M_\alpha}(x)) + 1$ by $S_{i,V}$. x only enters B_β (for $\beta \subset \alpha$) after stage $h(x, \Phi_{M_\alpha(x)}) + 1$ so x has not entered B_β at stage $\Phi_V(x)$. By our assumption that $S_{i,V}$ is to the left of α so $S_{i,V}$ would not commit to ($\sharp$) as x is a potential witness it can keep out. □

In particular, Lemma 5.3 shows that for every node α working on a Q_h outcome, the equations obtained from the c outcome only involves a fixed number of Z_i's.

The next lemma shows that if we have several equations obtained from the c outcome (of γ_i's) along some path and $\gamma_i \subseteq \alpha$ for all such γ_i's, then the reduced equation from these equations is true in R_α.

Lemma 2.6.5. *Let α be on the true path. If we obtain a reduced equation from equations obtained by the c outcome for nodes $\gamma \subseteq \alpha$ then the reduced equation is true of the intersections mentioned in the equation in R_α.*

Proof. Every equation obtained by a c outcome for γ is an equation in R_γ. As $R_\gamma \subseteq R_\alpha$, equations in R_γ are also equations in R_α. If we have true equations in R_α and we deduce an equation from it, the deduced equation is true in R_α. Thus, the reduced equation is true of the intersections mentioned in the reduced equation in R_α. □

The following lemma shows that if α is on the true path, in defining $G(\alpha)$, k always exists.

Lemma 2.6.6. *Let α be a node working on a Q_h requirement. Let $S_0, ..., S_i$ be a listing of all higher priority S-requirements. We cannot get an equation from the c outcome of α on the true path of the form $Y_{i_0} \cup \cdots \cup Y_{i_k} = B$ where $\{i_0, ..., i_k\} \subseteq \{0, ..., i\}$.*

Proof. We prove the lemma by induction on i.

For $i = 0$, suppose by contradiction that we could obtain such an equation, i.e. we obtain the equation in R_α, $Y_0 = B$. By our scheme of switching between X and Y, we must have also obtained the equation in R_γ: $X_0 = B$ for some $\gamma \subseteq \alpha$. At some stage, the satisfaction of some D_i requirement puts some element x into $B \cap R_\alpha \cap R_\gamma$ as requirements being satisfied by nodes on the true path after the two equations both appear only take witnesses from $R_\alpha \cap R_\gamma$.

However, this would mean that x enters both X_0 (from the equation in R_γ: $X_0 = B$ for some $\gamma \subseteq \alpha$) and Y_0 (from the equation in R_α: $Y_0 = B$ for some $\gamma \subseteq \alpha$). As X_0 and Y_0 are disjoint, we have obtained our contradiction.

For $i + 1$, suppose by contradiction that we could obtain such an equation. By our scheme of switching between X and Y, we obtain all equations of length $i + 2$ involving all combinations of $Z_0, ..., Z_{i+1}$ (from the c outcome), i.e. we have $Z_0 \cup ... \cup Z_i \cup X_{i+1} = B$ and $Z_0 \cup ... \cup Z_i \cup Y_{i+1} = B$ for every combination of values for $Z_0, ..., Z_i$. For a fixed combination of values, the two equations $Z_0 \cup ... \cup Z_i \cup X_{i+1} = B$ and $Z_0 \cup ... \cup Z_i \cup Y_{i+1} = B$ imply the reduced equation in R_α: $Z_0 \cup ... \cup Z_i = B$ as X_{i+1} and Y_{i+1} are disjoint. As we are considering all combinations of $Z_0, ..., Z_i$, we have our contradiction by inductive hypothesis. □

From Lemma 5.5, we see that we are always able to switch one of the Z_i's.

Lemma 2.6.7. *For every i, we only switch Z_i finitely many times.*

Proof. By induction on i. Let s be the least stage such that all of the higher priority Z_i's do not switch again. We prove that once Z_i switches, it cannot switch back again. Whenever a higher priority Z_j switches, we switch Z_i back to X_i so at stage s, Z_i is defined to be X_i. If Z_i never switches to Y_i, we are done. Otherwise, Z_i switches from X_i to Y_i. The only reason why Z_i would switch back to X_i would be because some smaller indexed set switched and thus it cannot switch back after stage s. □

Lemma 2.6.8. *On the true path, for every requirement, there is a node that works on it.*

Proof. It suffices to show that each Q_h strategy is not repeated infinitely often. Every Q_h strategy is only repeated when it has a c outcome. By Lemma 5.3, we obtain an equation only involving $S_{i,V}$ strategies that come from higher priority S_i's. Thus, whenever we have a c outcome, we have obtain an equation involving a fixed number of Z's: $Z_0, \cdots, Z_k$. By Lemma 5.5, we are always able to switch some Z from an X to a Y. By the previous lemma, we can only switch every Z_i finitely many times. Therefore, we cannot have a c outcome occur infinitely often and must go to the ∞ outcome. □

Lemma 2.6.9. *For all i, the D_i requirement is satisfied.*

Proof. This follows by Lemma 5.1 and similar reasoning as in the one split case. □

Lemma 2.6.10. *For all i, S_i is satisfied.*

Proof. If we never switch to a Y_i after all of the higher priority Z_j's have finished switching, we do not reset the $S_{i,V}$ strategies. If we do switch, we finish switching at some stage s by Lemma 5.6. We do not reset the S_i strategies again after we finish switching so there is a node working on $S_{i,V}$ for every V on the true path and thus the S_i requirement is satisfied. □

Lemma 2.6.11. *For all h, Q_h is satisfied.*

Proof. By the previous lemmas. Eventually the Z_i's stop switching and some α on the true path working on Q_h must get the ∞ outcome, i.e. there are infinitely many elements in M_α that are kept out of B until stage $h(x, \Phi_{M_\alpha}(x)) + 1$. □

□

2.7 Further generalizations and questions

One way to generalize Theorem 2.3 is to look at generalizations of being semilow (as semilow is equivalent to being nonspeedable). A particularly interesting generalization is that of the notion of semilow$_{1.5}$.

Definition 2.7.1. An r.e. set A is semilow$_{1.5}$ if and only if

$$\{e : W_e \cap \overline{A} \text{ infinite } \} \leq_1 \text{Inf}.$$

Semilow$_{1.5}$ sets occur when studying the lattice of r.e. sets. Maass [8] showed that if A is cofinite then A is semilow$_{1.5}$ if and only if $\mathcal{L}^*(A) \cong^{\text{eff}} \mathcal{E}^*$ (where $\mathcal{L}(A)$ is the lattice of r.e. supersets of A and $\mathcal{E}$ is the lattice of r.e. sets. The * denotes that we quotient out by the finite sets).

Semilow$_{1.5}$ sets also have a characterization using complexity theoretic notions closely related to nonspeedable sets. Instead of studying the property of having just one a.e. fastest program, Bennison and Soare [1] defined the notion of a *type 1 c.e. complexity sequence*, which is informally a sequence of lower bounds for all running times of programs for A (with some finite flexibility). They showed that a set has a type 1 c.e. complexity sequence if and only if it is semi-low$_{1.5}$.

One question to examine is whether we can replace semilow with semilow$_{1.5}$ in the statement of the main theorem. In fact, a stronger statement holds [6]. Maximal sets are not semilow$_{1.5}$ but have the property that if X and Y form a split of a maximal set and neither is recursive, then both X and Y are semilow$_{1.5}$. We give a brief proof for completeness.

Lemma 2.7.2. *For B maximal, if X and Y form a split of B (and neither is recursive), for every r.e. W, $W - X$ infinite if and only if $W \cap Y$ is infinite.*

$W \cap Y$ being infinite is a Π^0_2 property so by Lemma 6.2, X (and by symmetry, Y) is semilow$_{1.5}$.

Proof of Lemma 6.2. ($\Leftarrow$) is immediate as X and Y are disjoint. For the other direction, assume that $W - X$ is infinite for some r.e. W. By maximality, we must have $W \cap \overline{B}$ finite or $\overline{W} \cap \overline{B}$ finite. If we have $W \cap \overline{B}$ finite, then $W \cap Y$ is infinite as $W - X$ is infinite. If we have $\overline{W} \cap \overline{B}$ finite, we show that $W \cap Y$ cannot be finite by contradiction. Suppose that it were finite. Then the complement of Y is equal to $X \cup W \cup (\overline{W} \cap \overline{B})$ minus the finitely many elements in $W \cap Y$. As X and W are r.e. and $\overline{W} \cap \overline{B}$ is finite, the complement of Y is r.e. thus Y is recursive, contradicting the assumption that neither X nor Y is recursive. □

The following corollary follows immediately:

Corollary 2.7.3. *There is a non-semilow$_{1.5}$ set B such that if X and Y form a split of B then at least one of X or Y is semilow$_{1.5}$.*

A further generalization of the notion of being semilow is the notion of being semilow$_2$. We discuss further splitting questions related to semilow$_2$ and speedability in the next chapter.

Chapter 3

On semilow$_2$ sets

3.1 Overview

Let $V \searrow B$ denote $\{x : x \text{ enters } V \text{ and then enters } B\}$.

Definition 3.1.1. An r.e. set B is semilow$_2$ if $\{e : W_e \cap \overline{B} \text{ infinite}\} \leq_T \emptyset''$

The main theorem of this chapter is the following:

Theorem 3.1.2. *There exists a non-semilow$_2$ set B such that for all X, Y that form a split of B, either X or Y is semilow$_2$.*

To make B non-semilow$_2$, we diagonalize against all functions recursive in $\emptyset''$ that could witness the Turing reducibility. We only deal with X_i, Y_i such that X_i, Y_i is a split of B as it is Π_2 to determine whether X_i, Y_i form a split of B. Let Z_i denote the split we are trying to make semilow$_2$.

3.2 Requirements

First, we have the requirement to diagonalize against any function recursive in $\emptyset''$ in order to show that B is not semilow$_2$:

$$R_i : \Gamma_i \text{ is total and only takes values 0 or 1 } \rightarrow$$
$$(\exists e)(\Gamma_i(e, \emptyset'') = 0 \wedge W_e \cap \overline{B} \text{ infinite or } \Gamma_i(e, \emptyset'') = 1 \wedge W_e \cap \overline{B} \text{ finite })$$

For each R_i, we will build a set W^i to witness the incorrectness of Γ_i by keeping elements in W^i out of B until we see that Γ_i predicts that $W^i \cap \overline{B}$ is infinite. Once we see it predict $W^i \cap \overline{B}$ infinite, we will put all elements of W^i into B.

Observe that by the fixed point theorem, we have an index e_i for the W^i we are building.

We also have the requirement to make either X_i or Y_i semilow$_2$:

$$S_{i,V} : (\exists \Phi)\Phi^{\emptyset''}(e) = \begin{cases} 1 & \text{if } W_e \cap Z_i \text{ is infinite} \\ 0 & \text{if } W_e \cap Z_i \text{ is finite} \end{cases}$$

3.3 Priority Tree

Let $\Lambda = \{l < r < w < i < f < \text{blank}\}$. The priority tree is a subset of $\Lambda^{<\omega}$.

Each R_i node has 2 outcomes: l (for left) and r (for right). On the r outcome, we see that Γ_i converges and is equal to 0 so we start ensuring that $W_e \cap \overline{B}$ is finite by putting elements from W_e that appear into B. We will also redo all lower priority requirements. On the l outcome, we do not see that $\Gamma_i(e, \emptyset'') = 0$. Under this outcome, we will have child-nodes $R_{i,j}$. The $R_{i,j}$ node will attempt to keep one element out of B in an attempt to ensure that $W_e \cap \overline{B}$ infinite. We will arrange our priority tree so that after the k outcome, the first child-node $R_{i,0}$ will appear. We will refer to R_i as the parent node of $R_{i,j}$.

Each $R_{i,j}$ node will have two outcomes: l (the same as above) and r (the same as above). We will refer R_i as the parent node and the $R_{i,j}$ nodes as the child-nodes of R_i. We will also refer to any node working on a R_i or $R_{i,j}$ node as a R-node.

Each S_i-node will have its first child-node S_{i,V_0} as its immediate successor and has successor blank. The parent node S_i's purpose is to keep trap of the status of the child nodes and give a name to the witness W^i that it and the child nodes are building.

Each $S_{i,V}$ node will have two outcomes: w (for win), i (for infinite) and f (for finite). On the w outcome, we see infinitely many elements in $V \cap Y_i$. On the i outcome, we see that there are bigger and bigger finite sets in V that are not in B. On the f outcome, we do not see this (i.e. there are only less than k many many elements in V that are not in B at any time).

Under the i outcome, we have nodes $S_{i,V,e}$ that ensure that $W_j \cap \overline{B}$ is low for all W_j defined before the node working on $S_{i,V}$. These have a blank outcome. There are no further $S_{i,V}$ nodes under the i outcome.

Under the r outcome, we repeat all S_i requirements and child-nodes $S_{i,V}$ that appear between the parent node R_i and r. In the construction, we will have a link between the parent node and the f outcome, skipping all nodes in between.

We call nodes working on R_i or $R_{i,j}$ requirements R-nodes and the nodes working on S_i or $S_{i,V}$ requirements S-nodes.

3.4 Notation

In the following construction, we use the notion of "pool". Recall that each node on the tree uses the pool given by its predecessor. Let α be a node on the tree and let β be its predecessor. If α is a R-node, $P_\alpha = P_\beta$. If α is a node working on some $S_{i,V}$ requirement, α builds two sets, U_α (an approximation to V) and N_α (a new pool). If α has outcome w,

$P_\alpha = P_\beta$. If α has outcome i, $P_\alpha = V$. If α has outcome f, $P_\alpha = N_\alpha$. If α is not working on some $S_{i,V}$ requirement, $P_\alpha = P_\beta$.

Another notation that we use in the proof is the notion of used and unused elements. Elements are used once they are picked by nodes to go into their witness W. An element becomes unused when the construction goes to the left of the true path and resets everything to the right.

3.5 Intuition

Note that in the construction the R-requirements are satisfied by their child nodes - each of which only uses at most finitely many elements. Each child node either tries to restrain one more element or puts elements from their witness W (built by the parent node) into B.

To make A (or C) semilow$_2$, it suffices to have a Δ_3 way to determine whether or not $V_e \cap \overline{A}$ (or $V_e \cap \overline{C}$) is infinite. Suppose α is a node on the true path working on the S_{i,V_e} strategy. If $V_e \cap C$ is infinite, we are done as $V_e \cap C$ is a Π_2 property and we can conclude that $V \cap \overline{A}$ is infinite since A and C are disjoint. If $V_e \cap C$ is finite and $V \searrow B$ is always bounded by some fixed number k, then $V \cap \overline{B}$ is finite in the limit. Thus, $V_e \cap \overline{A}$ is finite.

Now, suppose that $V_e \cap C$ is finite and suppose that the cardinality of $V \searrow B$ increases infinitely often. If we build B in V for nodes after α, then most of the later "action" of nodes on the true path appearing after α will be inside V_e. As $V_e \cap C$ is finite, any element used later on the true path is going into A (if it does go into B). Now, we would like to conclude that C is semilow$_2$.

Let $W_0, ..., W_i$ be a listing of the sets we build for satisfying R-requirements that appear before α. Fix any V^*. If V^* is finite, we are done as being finite is a Σ_2 property. Now, assume that V^* is infinite. To answer the question: "Is $V^* \cap \overline{C}$ infinite?", it suffices to ask the equivalent question of whether $V^* \cap A$ is infinite or $(V^*\overline{C}) \cap W$ for one of the W's in our $W_0, ..., W_i$ listing is infinite.

This is because for elements in B, we know that A and C are disjoint so it suffices to ask the question: "Is $V^* \cap A$ infinite?" (a Π_2 question). By making $W_i \cap \overline{B}$ low, asking whether $(V^*\overline{C}) \cap W$ is infinite becomes a Π_2 question. The disjunction of a finite number of Π_2 statements is Π_2 and so answering whether $V^* \cap \overline{C}$ is infinite is Π_2.

3.6 Construction

Each node α has a pool P_β that is given to it by its predecessor, β. First, we find the approximation TP_s to the true path TP (the leftmost path travelled through infinitely often) at stage s and then we will let the nodes on TP_s act. The parent nodes α working on R_i will be building W_α to satisfy their requirement. Their child nodes will also be working on the same W_α.

Finding TP$_s$

Suppose that the n^{th} node α of TP_s for $n < s$ has been defined. We now define the next node of TP_s depending on which requirement α is working on.

R-node

If β is working on a $R_{i,j}$ requirement, we test whether $l \in \emptyset''$ for $l \leq j$ by asking whether $\phi_l(x) \downarrow \forall x \leq s$ and we estimate the value of $\Gamma_i(e, \emptyset'')$ according to this test.

1. (Estimating $\emptyset''$) If $\phi_l(x) \downarrow$ for all $x \leq s$, then we guess that $l \in \emptyset''$. If not, we guess that $l \notin \emptyset''$.

2. (Estimating the value of $\Gamma_i(e, \emptyset'')$) Calculate $\Gamma_i(e, \emptyset'')$ according to the guesses we obtained in (1).

3. (Diverges or does not equal to 0) If $\Gamma_i(e, \emptyset'')$ diverges or converges and is not equal to 0, let α be its l outcome.

4. (Converges and equals 0) Otherwise, go to the r outcome and create a link from β's parent node to f, skipping all nodes in between. This link is only removed when our current guess $\Gamma_i(e, \emptyset'')$ changes or if a higher priority R-node creates a link.

$S_{i,V}$-node

Let E_s^α be the set of used elements x at stage s such that x is used by a node $\gamma \leq_L \alpha$.

If $|\{x : x \in V_s \cap A_s\}| > \max_{0 \leq t < s}(|\{x : x \in V_t \cap A_t\}|)$, let $P_\alpha = P_\beta$ and go to the w outcome.

If $|\{x : (x \in E_s^\alpha \cap V_s \cap P_\beta) \wedge x \notin B_s\}| > \max_{0 \leq t < s}(|\{x : (x \in E_t^\alpha \cap V_t \cap P_\beta) \wedge x \notin B_t\}|)$, go to the b outcome and enumerate the elements in $\{x : (x \in E_s^\alpha \cap V_s \cap P_\beta) \wedge x \notin B_s\}$ into U_α. Let x be the least element in $\{x : (x \in E_s^\alpha \cap V_s \cap P_\beta) \wedge x \notin B_s\}$. Reset N_α. Put all elements not in U_α and not restrained by node $\gamma \leq_L \alpha$ into B. Let $P_\alpha = U_\alpha$.

Otherwise, go to the c outcome and add another unused number from P_β into N_α. Let $P_\alpha = N_\alpha$.

Action

Now let the nodes in TP_s act out according to their description below. If there is a link between nodes in TP_s, we do not carry out any actions for the nodes in between and go directly to the endpoint of the link.

Let α be the node we are working on.

R-node

If the next node of TP_s is α's f outcome or if there is a link from α to a f outcome of one of its child nodes, put all elements in W_α not being restrained by any node $\gamma \leq_L \alpha$ into B. Otherwise, ask whether x_α has been defined. If x_α has not been defined, pick an unused witness x in P_α that is not being restrained by any node $\beta \leq_L \alpha$ and let $x_\alpha = x$. Put x_α into W_γ (where γ is α's parent node) and keep x out of B. If x_α has been defined, continue keeping x_α out of B.

$S_{i,V}$-node

If our current guess at α's outcome is w or f, go to the next node. Otherwise, let x be the least such that $x \in |\{x : (x \text{ is unused and } x \in V_s \cap P_\beta) \wedge x \notin B_s\}$ and put all unused elements $\leq x$ in B.

$S_{i,V,e}$-node

(α is the parent node that initiates this process) Let $W_0, ..., W_k$ be a listing of W_β's for $\beta \subseteq \alpha$. For each $j = 0, ..., k$, ask whether $\{e\}_s^{W_j \cap \overline{B}}(e)$ converges. If so, define $r(\alpha, j)$ to be its use. If not, define $r(\alpha, j)$ to be 0. Restrain lower priority nodes from using numbers less than $r(\alpha, j)$ for $j = 0, ..., k$. Ask whether $\{e\}_s^{U_\alpha \cap \overline{B}}(e)$ converges. If so, define $r(\alpha, e)$ to be its use. If not, define $r(\alpha, e)$ to be 0. Restrain lower priority nodes from using numbers less than $r(\alpha, e)$.

3.7 Verification

Recall that the true path denotes the leftmost path travelled through infinitely often in the construction. β will be α's predecessor unless otherwise stated.

Lemma 3.7.1. *Let α be on the true path. Either there exists some stage s such that P_α is infinite from stage s onwards or there are infinitely many stages where elements are being added to P_α.*

Proof. The proof is by induction on α. Let α be a node working on a $S_{i,V,e}$-strategy. By induction, P_β is either infinite from some stage s or there are infinitely many stages when elements are added to P_β so if $P_\alpha = P_\beta$, the lemma immediately follows. If the true outcome of α is i, then $P_\alpha = U_\alpha$ and we only go to i when we can add elements to U_α so the lemma follows. If the true outcome of α is f, we do not go to the left of the true path from some stage onwards and thus N_α does not reset after some stage s. We add a new element into N_α every time we go to the f outcome and as $P_\alpha = N_\alpha$, the lemma follows. □

Lemma 3.7.2. *If α is the node on the true path working on the $S_{i,V}$ requirement such that the true outcome is that $V \searrow B$ expands infinitely often and we do not build a link in the*

construction that skips over α infinitely often, let $W_0, ..., W_k$ be a listing of witnesses that R-nodes above α build. Then for all $j = 0, ..., k$, $W_j \cap \overline{B}$ is low.

Proof. Fix i, V and α such that they satisfy the hypothesis of the lemma and let $W_0, ..., W_k$ be a listing of witnesses that R-nodes above α build. Fix some j and some e. Let $Z = W_j \cap \overline{B}$. We would like to show that if $\{e\}_s^Z(e) \downarrow$ for infinitely many stages then $\{e\}^Z(e) \downarrow$. Assume that $\{e\}_s^Z(e) \downarrow$ for infinitely many stages. By hypothesis, there is a node γ on the true path that works on $S_{i,V,e}$ and let s be the stage that the construction does not go to the left of γ. By assumption, there is a stage $t > s$ such that $\{e\}_t^Z(e) \downarrow$ and the construction goes to γ at stage t. By construction, γ restrains elements from going in below the use of $\{e\}_t^Z(e)$. As the construction does not go to the left after stage t, γ's restraint is there from stage t onwards and no node can put in an element below its restraint. Thus, $\{e\}^Z(e) \downarrow$. As e was arbitrary, $W_j \cap \overline{B}$ is low. □

Lemma 3.7.3. *For every i, R_i is satisfied.*

Proof. Let s be the stage that the construction does not go to the left of η where η is the last node on the true path working on the R_i requirement. Either $\Gamma_i(e, \emptyset'')$ converges and equals 0 or it diverges or it converges and equals 1. If $\Gamma_i(e, \emptyset'')$ diverges, we are done.

If $\Gamma_i(e, \emptyset'')$ converges and equals 1, we eventually see on the tree that $\Gamma_i(e, \emptyset'')$ does not equal 0 and after this point, we go to the l outcome. Thus by construction, there are infinitely many child-nodes of R_i on the true path. All of them have the l outcome so all of them are trying to keep one more element out of B. Let α be any child-node of R_i on the true path. As α restrains its witness x out of B, only nodes above α or to the left are able to put x into B or reset α's witness. The construction travels to the left of α finitely often. Let t be the stage after which the construction does not go to the left of α. Then after stage t, α's witness is not reset or put into B so α successfully restrains an element out of B. As child-nodes of R_i always pick unused elements as their witnesses, each child-node ensures that one more element in W is not in B. Therefore, $W_e \cap \overline{B}$ is infinite.

If $\Gamma_i(e, \emptyset'')$ converges and equals 0, we eventually see on the tree that $\Gamma_i(e, \emptyset'')$ equals 1 and will either have a link from η to the f outcome or η will have the f outcome. From some point on, every element in W (minus finitely many restricted by nodes to the left) is put into B so $W_e \cap \overline{B}$ is finite. □

Lemma 3.7.4. *For every i, S_i is satisfied.*

Proof. Let η be the last S_i node on TP and let s be the first stage that the true path does not go to the left of η. To satisfy S_i, it suffices to have, for all V, a Δ_3 method of determining whether $V \cap \overline{A}$ (or $V \cap \overline{C}$ if an i outcome appears on the true path) is infinite or not.

Suppose that there is no i outcome after η on TP. We would like to argue that A is semilow$_2$ by arguing that there is a series of Δ_3 questions whose disjunction is equivalent to asking whether $V \cap \overline{A}$ is infinite. Fix some arbitrary infinite V and let α be the node on TP that is working on $S_{i,V}$. If α has outcome w on the true path, then $V \cap C$ is infinite and thus $V \cap \overline{A}$ is infinite. As $V \cap C$ being infinite is a Π_2 property, we are done.

Now, suppose that α has the f outcome. Then we have a few possibilities: 1) there is some k such that at any stage, there are only less than k many numbers in V and out of B or 2) elements in $V \searrow B$ are i) used by elements to the left ii) used by $W_0, ..., W_i$ or iii) are not in the pool or iv) are in B.

In the first case, there are only finitely many numbers in V and out of B in the limit. Thus minus finitely many numbers, $V \subseteq B$ and thus it suffices to ask whether $V \cap C$ is infinite or not (a Π_2 question). If it is infinite, $V \cap \overline{A}$ is infinite. If not, $V \cap \overline{A}$ is finite. The same holds for 2) iv).

There are only finitely many elements in 2) i) so α cannot have an f outcome with only 2) i) holding. If 2) iii) held, we know that either these elements are in the other pool at an earlier $S_{i,V}$-node γ pool split or in B by the action of the construction (4.1.2). If they are in the other pool, then either they are in U_α or U_γ, but both U_α and U_γ are finite as the true outcome of the either node is not i. The elements cannot be in N_γ and not some other U_δ as either $P_\beta \subseteq N_\gamma$ or N_γ is reset infinitely often (as we know that the true path goes to the left of the f outcome of γ by construction of pools).

If 2) ii) held, we know that all elements in V that are out of B are in $W_0, ...W_i$. As $W_j \cap \overline{B}$ is low by Lemma 6.2, it suffices to ask whether there is some $j = 0, ...i$ such that $W_j \cap \overline{B} \cap V$ is infinite (a Π_2 question).

Thus it suffices to ask the disjunction of "Is $V \cap C$ infinite?" and $W_j \cap \overline{B} \cap V$ (for $j = 0, ..., i$), which is a Π_2 question and thus is Δ_3.

Now suppose that there is an i outcome after η on TP. We would like to conclude that C is semilow$_2$. Let α with requirement $S_{i,V}$ be the child-node with the i outcome after η on TP. We only have the i outcome when $V \cap C$ is finite and thus all elements in V that go into B are going into A. As discussed in the intuition section, we have that $V^* \cap \overline{C}$ if and only if $V^* \cap A$ is infinite or $V^* \cap V$ is infinite or $W_j \cap \overline{B} \cap V^*$ is infinite for the $W_0, ...W_i$ of higher priority than α. This holds because it suffices to ask whether $V^* \cap A$ is infinite for elements in B since A and C are disjoint. The other elements used in the construction (not in B) are those used by strategies to the right of α that are put in $W_0, ..., W_i$ because elements not in V and not in P_β and not used (in the limit) by the construction for R-nodes are put into B and elements used by the strategies below α below outcome i are all in V. As we make $W_j \cap \overline{B}$ is low so asking $W_j \cap \overline{B} \cap V^*$ is a Δ_3 question. $\square$

Chapter 4

Recursively avoiding reals

4.1 Introduction

It is folklore that if A is r.e. and not recursive, there exists a set X of the same degree as A such that X is not contained in any countable Π^0_1 class. In this chapter, we prove a stronger theorem using the definition of rec avoiding that we will introduce below. Its proof is simpler than the usual proof of this folklore fact as rec avoiding is a Σ^0_2 definition while X not being contained in any countable Π^0_1 class is a Π^0_2 statement.

Recall that being ranked means that x belongs to some Π^0_1 P and is thrown out at some step n of the Cantor-Bendixon analysis of P. The Cantor-Bendixon analysis throws away points until we are left with only isolated points, which we call the perfect kernel $D(P)$. $P \backslash D(P)$ is a countable set. Any countable set has a recursive element. If P is a Π^0_1 class, then the intersection of P with a countable set is a countable set so its intersection must have a recursive path. If x is not ranked then it is not contained in any countable class. However, this does not mean that it is in a class with no recursive element. Suppose x is not ranked but belongs to a Π^0_1 class P with some ranked element y. Let Q be some Π^0_1 class such that Q witnesses that y is ranked. $P \cap (Q \backslash D(Q))$ is a countable set that is nonempty (as it contains y) and must have a recursive element.

Motivated by this, we give the following definition:

Definition 4.1.1. For a real x, we say that x is *rec avoiding* (for recursively avoiding) if and only if there is some Π^0_1 class P such that $x \in P$ and P does not contain a recursive element.

Rec avoiding is stronger than being not ranked as being rec avoiding implies being not ranked. Suppose otherwise. Let x be rec avoiding and ranked. Then x belongs to some Π^0_1 class Q such that $Q \backslash D(Q)$ contains x. Let P be such that P witnesses that x is rec avoiding. Then $P \cap (Q \backslash D(Q))$ is countable and nonempty and thus it contains a recursive element, contradicting our assumption that P does not contain a recursive element.

4.2 Existence of rec avoiding reals in every Δ^0_2 degree

In this section, we prove the following theorem:

Theorem 4.2.1. *In every nonrecursive Δ^0_2 degree, there exists a rec avoiding x.*

Proof. Let **a** be a nonrecursive Δ^0_2 degree. We construct a recursive tree T such that the set of infinite paths through T is a Π^0_1 class P with no recursive element and P contains an element of degree **a**.

Let X be some set of degree **a**. We enumerate Φ such that $Z = \Phi(X)$ and compute T such that $Z \in [T]$.

Requirement 1: If ν extends σ, $\Phi(\nu)$ extends $\Phi(\sigma)$.

Requirement 1 is necessary for $\Phi(X)$ to be a path in T.

Requirement 2: If σ_1 is incompatible with σ_2 and both are in the domain of Φ, then $\Phi(\sigma_1)$ is incompatible with $\Phi(\sigma_2)$.

Requirement 2 is necessary for us to compute X from Z to conclude that Z has the same degree as X.

Construction

We label a node σ as *extendible* if σ is equal to $\Phi(\tau)$ for some τ. Let l_s be the highest level at which there is a node in T of level l_s at stage s.

By the Limit Lemma and since X has Δ^0_2 degree, there is a recursive sequence $\{f_s\}_{s\in\omega}$ such that the characteristic function of X is the limit of the f_s. We can assume without loss of generality that the range of f_s is a subset of $\{0, 1\}$ for all s.

At stage 0, we do nothing.

At stage $s + 1$, we do the following:

1. (Extend Φ) Look at the current approximation to X, f_s, and compute f_s up to $s - 1$. Let $\sigma = \langle f_s(0), ..., f_s(s - 1)\rangle$. Let τ be of least length such that σ extends τ and $\Phi(\tau)$ is not defined and let τ^- be $\tau \upharpoonright (|\tau| - 1)$.

   Define $\Phi(\tau)$ to extend $\Phi(\tau^-)$ (to satisfy Requirement 2 and to extend to level l_s) by taking $\Phi(\tau)$ to be the leftmost finite sequence of length l_s that extends $\Phi(\tau^-)$ such that if $\Phi(\nu)$ is defined and ν is incompatible with τ, then $\Phi(\tau)$ is incompatible with $\Phi(\nu)$.

   Put $\Phi(\tau)^\frown 0$ and $\Phi(\tau)^\frown 1$ in T.

2. (Extend all extendible nodes to satisfy Requirement 1) For every $\nu \neq \tau$ and in T extending an extendible node, if ν' is an extension of ν of length l_s and is DNR (diagonally nonrecursive) for all values between the length of ν and l_s, put ν' into T.

Verification

In step (1), we are able to define $\Phi(\tau)$ by the following reasoning and induction: Look at the first place of disagreement between τ and ν. If this place occurs in the domain of τ^-, $\Phi(\tau^-)$ is incompatible with $\Phi(\nu)$ by induction and definition and any extension of $\Phi(\tau^-)$ is thus incompatible with $\Phi(\nu)$. Thus, we can assume that $\tau = (\tau^-)^\frown i$ and ν extends $(\tau^-)^\frown |1-i|$ where $i = 0, 1$ and $|1-i|$ denotes the absolute value of $1-i$. We can then take $\Phi(\tau^-)$ to be the leftmost extension of $\Phi(\tau)^\frown 1$ up to level l_s as $\Phi(\nu)$ must have extended $\Phi(\tau^-)^\frown 0$ by construction.

By construction, Requirement 1 is met and thus $\Phi(X) = Z$ is in $[T]$. Now we claim that Z computes X. Suppose that we have already computed $\sigma = X$ restricted up to length n. To compute the value of the characteristic function of X at value n, we enumerate Z and Φ until either $\Phi(\sigma^\frown 0) \subseteq \tau$ or $\Phi(\sigma^\frown 1) \subseteq \tau$ for some $\tau \subseteq Z$. One of the two must occur as $\Phi(X) = Z$ and only one of the disjuncts holds by Requirement 2, which holds by our way of defining $\Phi(\tau)$. Whichever one holds is the correct value of X since $\Phi(X) = Z$.

If $f \in [T]$ is not Z then the approximation f_s stops going through f. Let n be the least such that the construction does not define $\Phi(\tau)$ to be $f \upharpoonright n$. By step (2), f is DNR for all values greater than n and thus all $f \in [T]$ such that $f \neq Z$ are not recursive. As X is not recursive and Z computes X, Z is not recursive and so $[T]$ has no recursive member. □

Observe that for the Π^0_1 class P constructed in the proof, all elements except for the element of specified Δ^0_2 degree are DNR mod finite and thus have DNR degree. As the $\{0,1\}-$DNR degrees are the same as complete extensions of Peano Arithmetic, the proof of our theorem gives the following:

Corollary 4.2.2. *For any Δ^0_2 degree* $\mathbf{a}$*, there exists a Π^0_1 class P such that there is an element x of P with degree* $\mathbf{a}$ *and whose elements all have PA degree except possibly x.*

As we said in the introduction, Theorem 5.2 also implies that if A is r.e. and not recursive, there exists a set X of the same degree as A such that X is not contained in any countable Π^0_1 class as we can take X to be as constructed in the proof of Theorem 1 and X cannot be contained in any countable Π^0_1 class Q or else $P \cap Q$ would contain a recursive element for P witnessing that X is rec avoiding.

Further observations

The previous theorem showed that there is a rec avoiding real in every nonrecursive Δ^0_2 degree. We now give an example of x's that are not rec avoiding. The canonical code of P is the coding of P by its representative tree with no dead ends.

Theorem 4.2.3. *Suppose that G is 2-generic and $\Phi(G)$ belongs to a Π^0_1 class $P = [T]$. Then there is a closed subset of P whose canonical code is r.e. In particular, P has a recursive element. If $\Phi(G)$ is not recursive, it is contained in a perfect closed subset of P whose canonical code is r.e.*

Proof. Since $\Phi(G)$ belongs to $[T]$, there is some p such that $p \Vdash G \in [T]$. Thus, $p \Vdash \Phi(G)$ is total and $p \Vdash \forall n(\Phi(G) \upharpoonright n \in T)$. Find recursive $p = p_0 > p_1 > p_2 > ...$ such that p_i decides $\Phi(G) \upharpoonright i$, i.e. $\Phi(p_i) \upharpoonright i$ is defined with use less than the length of p_i. Let $X = \lim_{i\to\infty} \Phi(p_i)$. X is recursive and belongs to $[T] = P$.

For the closed subset of P whose canonical code is r.e., take the paths through $T^* = \{\sigma : (\exists q < p)\Phi(q) = \sigma\}$. The leftmost path in T^* is a recursive element in P. If $\Phi(G)$ is not recursive, take p to also force that $\Phi(G)$ is not recursive. T^* is a perfect tree. □

Bibliography

[1] Victor L. Bennison and Robert I. Soare. "Some lowness properties and computational complexity sequences". In: *Theoret. Comput. Sci.* 6.3 (1978), pp. 233–254. ISSN: 0304-3975.

[2] M. Blum and I. Marques. "On complexity properties of recursively enumerable sets". In: *J. Symbolic Logic* 38 (1973), pp. 579–593. ISSN: 0022-4812.

[3] S. Barry Cooper and Sergey S. Goncharov, eds. *Computability and models*. The University Series in Mathematics. Perspectives east and west. Kluwer Academic/Plenum Publishers, New York, 2003, pp. xx+375. ISBN: 0-306-47400-X.

[4] R. G. Downey and Richard A. Shore. "Splitting theorems and the jump operator". In: *Ann. Pure Appl. Logic* 94.1-3 (1998). Conference on Computability Theory (Oberwolfach, 1996), pp. 45–52. ISSN: 0168-0072. DOI: 10.1016/S0168-0072(97)00066-3. URL: http://dx.doi.org/10.1016/S0168-0072(97)00066-3.

[5] R. G. Downey and L. V. Welch. "Splitting properties of r.e. sets and degrees". In: *J. Symbolic Logic* 51.1 (1986), pp. 88–109. ISSN: 0022-4812. DOI: 10.2307/2273946. URL: http://dx.doi.org/10.2307/2273946.

[6] Rod Downey and Michael Stob. "Splitting theorems in recursion theory". In: *Ann. Pure Appl. Logic* 65.1 (1993), p. 106. ISSN: 0168-0072. DOI: 10.1016/0168-0072(93)90234-5. URL: http://dx.doi.org/10.1016/0168-0072(93)90234-5.

[7] Michael A. Jahn. "Implicit measurements of dynamic complexity properties and splittings of speedable sets". In: *J. Symbolic Logic* 64.3 (1999), pp. 1037–1064. ISSN: 0022-4812. DOI: 10.2307/2586618. URL: http://dx.doi.org/10.2307/2586618.

[8] Wolfgang Maass. "Characterization of recursively enumerable sets with supersets effectively isomorphic to all recursively enumerable sets". In: *Trans. Amer. Math. Soc.* 279.1 (1983), pp. 311–336. ISSN: 0002-9947. DOI: 10.2307/1999387. URL: http://dx.doi.org/10.2307/1999387.

[9] Roland Sh. Omanadze. "Splittings of effectively speedable sets and effectively levelable sets". In: *J. Symbolic Logic* 69.1 (2004), pp. 143–158. ISSN: 0022-4812. DOI: 10.2178/jsl/1080938833. URL: http://dx.doi.org/10.2178/jsl/1080938833.

[10] Robert I. Soare. "Computational complexity, speedable and levelable sets". In: *J. Symbolic Logic* 42.4 (1977), pp. 545–563. ISSN: 0022-4812.

[11] Robert I. Soare. *Recursively enumerable sets and degrees*. Perspectives in Mathematical Logic. A study of computable functions and computably generated sets. Springer-Verlag, Berlin, 1987, pp. xviii+437. ISBN: 3-540-15299-7.