Title
Lowness For Computational Speed

Author
Bayer, Robertson Edward

Publication Date
2012

Peer reviewed|Thesis/dissertation

Lowness For Computational Speed

by

Robertson Edward Bayer

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 Theodore Slaman, Chair
Professor Leo Harrington
Professor Alistair Sinclair

Fall 2012

Lowness For Computational Speed

Abstract

Lowness For Computational Speed

by

Robertson Edward Bayer

Doctor of Philosophy in Mathematics

University of California, Berkeley

Professor Theodore Slaman, Chair

From the original definition of a set whose jump is as simple as possible ($A' \equiv_T 0'$), to more recent definitions involving randomness, notions of lowness appear throughout recursion theory. In that spirit, a non-recursive set A will be said to be *low for speed* if for any recursive set R and any computation of R from A, there is an oracle-free computation of R that is no more than polynomial-time slower than the A-computation.

We will construct such an r.e. set and discuss some properties of these sets. We will show that promptly simple r.e. sets cannot be low for speed, and also that there are non-prompt sets that are not low for speed. We conclude by showing that generic sets are low for speed if and only if $\mathcal{P} = \mathcal{NP}$.

Contents

Acknowledgments

First and foremost, I would like to thank my advisor Ted Slaman for his unending patience, support, guidance, and insight over the last five years. I cannot thank you enough for the countless enlightening and enjoyable conversations, mathematical and otherwise, we have had and for your incredible support.

Second, I give my heartfelt thanks to all the department staff that keeps this place running. In particular, I owe a substantial debt of gratitude to Barbara Peavy for providing me with interesting, diverse, and challenging teaching assignments and for being an endless resource of teaching knowledge. My time in the classroom has been one of the true highlights of my Berkeley experience, and I cannot imagine this place without you.

Finally, I would like to give my thanks to all the people without whom I never would have made it to Berkeley in the first place. To Mrs. Carol Vickery, who in junior high made it cool to be smart and started me down this road. To Profs. William Hrusa, Rami Grossberg, and John Mackey from Carnegie Mellon, who were my inspirations to pursue mathematics. And of course to my mother, who has always believed in me and supported me in everything I do.

Chapter 1

Preliminaries

1.1 Introduction

Various notions of lowness appear throughout recursion theory and computable mathematics. While there is no unifying rule for what gets called a lowness notion, the general idea is that low objects should have very little information in regards to the desired attribute. For example, an r.e. set A is called low_n if $A^n \equiv_T 0^n$; that is, its n^{th} jump is as simple as possible. More recently, Zambella defined a notion of lowness in relation to randoms by saying that a set A is low for random iff each Martin-Löf random real remains Martin-Löf random relative to A; that is, A is not useful in finding patterns in random reals [21]. The existence of such sets was proven by Kučera and Terwijn [9].

Similarly, computability theory and complexity theory have long been concerned with the effect, if any, of allowing computations access to extra information. Often, these questions ask whether adding non-recursive sets actually buys any interesting help with recursive sets. For example, as far back as 1950, Turing asked whether adding access to a source of randomness would be of any benefit to search algorithms [19]. This can be thought of as asking whether random sets are in some sense "low" for search speed. Over the years, complexity theorists have studied many aspects of randomness. Perhaps the most famous examples of this are the classical complexity theory questions of whether $\mathcal{P} = \mathcal{RP}$ and whether $\mathcal{P} = \mathcal{BPP}$. Again, this can be thought of as asking whether randomness is in some sense "low" for speed of computation. Even the famous question of whether $\mathcal{P} = \mathcal{NP}$ can be thought of as asking whether non-determinism is "low" for polynomial-time computability.

In terms of more classical mathematical logic, we can look to Gödel for an example. In 1936, he hypothesized that for any recursive function h there were infinitely many formulas provable in first order arithmetic whose proofs in second order arithmetic were shorter by a factor of h than the shortest proofs of them in first order arithmetic [6]. That is, if l is the length of the shortest proof in second-order arithmetic, then the shortest first-order proof has length at least $h(l)$. This conjecture was eventually proved by Parikh [16] in 1973. In the language of lowness, this could be phrased as saying that second order arithmetic is not

low for proof length over first order arithmetic.

In private correspondence, Eric Allender recently asked whether there was any non-recursive set A such that any recursive set computable in polynomial time from A must in fact be computable in polynomial time without A. A positive answer was given by Lance Fortnow, also in unpublished correspondence.

A set with the property that $\mathcal{P}^A \cap Rec = \mathcal{P}$ can in some sense be thought of as "low for polynomial speed." In that light, we will define a new notion of lowness, that of *low for speed*, which is more general and is not restricted to polynomial time computation and discuss various properties of these sets. Our main idea is that a set A is low for speed if and only if it provides no non-trivial speedup to computations of recursive sets. That is, knowing A does not provide any help in quickly computing recursive sets. Thus, unlike in typical recursion theory, we will be critically concerned with the number of steps required in various computations, not just with whether they give the correct answer.

In the remainder of this chapter, we give some background on our notation and describe our model of computation before giving the formal definition of low for speed and giving some general comments about the notion. We will also briefly discuss other lowness notions related to speed of computation.

In chapter two, we will explore sets which are extremely not low for speed and then construct an r.e. set which is low for speed. The construction will be given as a tree priority argument. Steffen Lempp [11] has produced an excellent set of notes providing an introduction to priority arguments for the interested reader.

Chapter three explores the connection between prompt simplicity and low for speed. The main results will be showing that no promptly permitting r.e. set can be low for speed, but that there are also r.e. sets which are not promptly permitting and yet fail to be low for speed in a strong way. The construction of this latter result will also be a priority argument.

Finally, in the fourth chapter we will discuss whether sets which are low for speed can be constructed which are also generic. We will not answer this question, but will instead show that answering it is equivalent to deciding the classical complexity theory question of whether $\mathcal{P} = \mathcal{NP}$.

Chapters two through four are each largely self-contained, relying only on the notation, conventions, and definitions given in this chapter. For clarity, definitions and concepts that are used in only one chapter are generally discussed at the beginning of the chapter in which they are used.

1.2 Notation and conventions

Before turning to the formal definition, it is helpful to formalize our notation and our model of computation.

Unless otherwise noted, our sets will be subsets of $2^{<\omega}$. Lower case Greek letters (σ, τ, etc) will be used to denote elements of $2^{<\omega}$ and for any $\sigma \in 2^{<\omega}$, we will use $|\sigma|$ to denote the length of σ. For $\sigma, \tau \in 2^{<\omega}$, we will use $\sigma \hat{} \tau$ to denote the string consisting of σ followed by τ,

and we will write $\sigma \prec \tau$ if σ is an initial segment of τ. Lower case Latin letters (n, m, s, k, l, etc) will be used for elements of $\mathbb{N}$. Abusing notation, if $n \in \mathbb{N}$, we will use $|n|$ to denote $\lceil \log_2(n) \rceil$, the number of bits needed to represent n.

When $A \subseteq 2^{<\omega}$ and $n \in \mathbb{N}$, we use $A \upharpoonright n$ to denote the set $\{\sigma \in A : |\sigma| \leq n\}$. When $A \subseteq \mathbb{N}$, we mean the set $\{m \in A : m < n\}$. The complement of A will be denoted $\overline{A}$.

Because we will be concerned not just with what sets are computable from others, but also with *how fast* those computations are, it is important to be concrete about our model of computation, and we will use the standard multi-tape Turing machine. For oracle machines, we will use the query machine as described by Cook [4]. Namely, a query machine is a standard Turing machine that in addition to the standard tapes and control features also contains a distinguished query tape and three distinguished states: the query state, the yes state, and the no state. When the machine enters the query state, it then transitions to the yes state or no state depending on whether the string currently written on the query tape is a member of the oracle. When counting steps required in a computation, a query to the oracle will count as a single atomic operation.

Other models of oracle computation are also common, and while all have the same computational power from a recursion theory standpoint, they can differ wildly in the time bounds on computations using them. For example, the model that consists of a read-only tape consisting of the characteristic function for the oracle has the property that determining whether a string of length n belongs to the oracle requires $O(2^n)$ steps since such a string would be recorded in the 2^nth position of the characteristic function. This would mean that computing $A \subseteq 2^{<\omega}$ from itself in the obvious way runs in time $O(2^n)$. The query machine model, however, counts this computation as running in time $O(n)$ since the machine needs only to copy the input to the query tape and then make a query.

Uppercase Greek letters (Φ, Ψ, Θ, etc) will be used to denote functionals and will be assumed to use machines as described above. We will write $\Phi(\sigma) = 1$ if Φ accepts σ, $\Phi(\sigma) = 0$ if it rejects it, and $\Phi(\sigma) \uparrow$ if Φ fails to converge. Again abusing notation, we will often use Φ to refer to both the functional/machine and the set computed by that functional. Relativizations using the oracle A will be denoted Φ^A, etc and are done using the query machines described above.

For any functional Φ and any $\sigma \in 2^{<\omega}$, we will use $t(\Phi, \sigma)$ to denote the number of steps used in the computation of $\Phi(\sigma)$. If $\Phi(\sigma) \uparrow$, we will say $t(\Phi, \sigma) = \infty$. For any $A \subseteq 2^{<\omega}$, $t(\Phi^A, \sigma)$ will be the number of steps in the oracle machine computation of $\Phi^A(\sigma)$ where queries to the oracle are considered atomic operations as previously described.

For an oracle machine Φ^A, we will also define the *use of* A in the computation $\Phi^A(\sigma)$, denoted $\mu(\Phi^A, \sigma)$, to be the length of the longest string written on the query tape in the computation of $\Phi^A(\sigma)$. If $\Phi^A(\sigma) \uparrow$, the use will be undefined.

In order to simplify some theorems and formulas, we impose the reasonable requirement that machines must at least read their input, and so $t(\Phi, \sigma) \geq |\sigma|$. As a byproduct of our model of computation, we get that

$$\mu(\Phi^A, \sigma) \leq t(\Phi^A, \sigma) \tag{1.1}$$

We will often be concerned with polynomial-bounded computation. To that end, we make the implicit assumption that all polynomials discussed are non-decreasing and positive. In fact, in most cases it is sufficient to consider only polynomials of the form $p(n) = n^c$ for some constant $c \in \mathbb{N}$.

We let $\mathcal{P}$ denote the set of all $A \subseteq 2^{<\omega}$ which are recursive and computable in polynomial time. That is, $A \in \mathcal{P}$ if and only if there is some total recursive Φ and some polynomial p such that $\Phi = A$ and $t(\Phi, \sigma) \leq p(|\sigma|)$ for all $\sigma \in 2^{<\omega}$.

As is standard, we let $\{W_e\}_{e\in\omega}$ be the standard enumeration of all r.e. sets. Namely, if $\{\Phi_e\}$ is an enumeration of all partial functionals, then we say $n \in W_e \leftrightarrow \Phi_e(n)\downarrow$.

Given any r.e. set U, we will use U_s for the stage s approximation to U in the standard sense. That is, we assume $\{U_s\}_{s\in\omega}$ is an increasing sequence of finite sets whose union is U. We will say n *enters* U *at stage* s if $n \in U_s - U_{s-1}$.

1.3 Low for speed

Before giving the definition of low for speed, we will consider the original question of Eric Allender and give Fortnow's proof of it. Before proving this theorem, recall that an r.e. set $A \subseteq 2^{<\omega}$ is hypersimple (Post, [17]) iff A is co-infinite and there is no recursive enumeration of disjoint finite sets D_n such that $\overline{A} \cap D_n \neq \emptyset$ for all n.

Theorem 1.1 (Fortnow)**.** *If A is hypersimple, B is recursive, and $B \in \mathcal{P}^A$, then $B \in \mathcal{P}$.*

Proof. Since A is hypersimple, by [18] there is no recursive function f such that

$$|\overline{A} \upharpoonright f(n)| \geq n \text{ for all } n \tag{1.2}$$

Suppose B is recursive and $B \in \mathcal{P}^A - \mathcal{P}$. Note that since $A \equiv_T^P \overline{A}$, $B \in \mathcal{P}^{\overline{A}}$ and so we can find a functional Ψ and a polynomial p such that $\Psi^{\overline{A}} = B$ and $t(\Psi^{\overline{A}}, \sigma) \leq p(|\sigma|)$. Our goal will be to construct a recursive function satisfying equation 1.2 contradicting the hypersimplicity of A.

For each n, let $D_n \subseteq 2^{<\omega}$ be a minimum-sized set of strings such that

$$\forall \sigma \ \ |\sigma| < n \Rightarrow \left[\Psi^{D_n}(\sigma)\downarrow = B(\sigma) \ \ \& \ \ t(\Psi^{D_n}, \sigma) \leq p(|\sigma|)\right] \tag{1.3}$$

Note that D_n is a recursive sequence of finite sets. From the definition, it is clear that $|D_n|$ is a non-decreasing sequence. Because an oracle computation taking at most $p(n)$ steps cannot query any strings of length exceeding $p(n)$, we also get that $|D_n| \leq |\overline{A} \upharpoonright p(n)|$.

Lemma 1.2. *The sequence $|D_n|$ is unbounded.*

Proof. Suppose not, and let n_0, k be such that $|D_n| = k$ whenever $n \geq n_0$. Suppose $n > n_0$. By equation 1.1 and the definition of D_n, we know $\mu(\Psi^{D_n}, \sigma) \leq p(n_0)$ whenever $|\sigma| < n_0$ so we have

$$\forall \sigma \ \ |\sigma| < n_0 \Rightarrow \left[\Psi^{D_n \upharpoonright p(n_0)}(\sigma)\downarrow = B(\sigma) \ \ \& \ \ t(\Psi^{D_n \upharpoonright p(n_0)}, \sigma) \leq p(|\sigma|)\right] \tag{1.4}$$

By the minimality of the size of D_{n_0}, we know $|D_n \upharpoonright p(n_0)| \geq |D_{n_0}| = k$. Since $|D_n| = k$, we must have $D_n = D_n \upharpoonright p(n_0)$. There are only finitely many possible sets of strings with lengths not exceeding $p(n_0)$, so one such subset F must appear infinitely often in the D_n sequence. Then $B = \Psi^F$, so $B \in \mathcal{P}$ since F is finite and Ψ^F runs in polynomial time.

□

Let $f(k) = p(n)$ for the smallest n such that $|D_n| \geq k$. Note that f is computable and that $|\overline{A} \upharpoonright f(k)| \geq |D_n| \geq k$, contradicting the hypersimplicity of A and giving our desired result.

□

Analyzing the proof above, we see that there is nothing special about the function $p(n)$ being a polynomial. In truth, we only needed $p(n)$ to be a computable bound on $t(\Psi^A, \sigma)$ so that the D_n sequence is computable. By following the same proof as above, we can get:

Corollary 1.3 (Of the proof of Theorem 1.1)**.** *Let A be hypersimple, R be recursive, and Ψ be a functional such that $\Psi^A = R$. If there is a total, non-decreasing, recursive function f such that $t(\Psi^A, \sigma) \leq f(|\sigma|)$, then there is a total Θ such that $\Theta = R$ and $t(\Theta, \sigma) \leq p(f(|\sigma|))$ for some polynomial p.*

We turn now to the formal definition of low for speed. Recall that our goal is to formalize the idea of a set being of no non-trivial help when computing recursive sets. The following definition is motivated by the original question asked by Eric Allender, the main difference being that we no longer restrict ourselves to the complexity class $\mathcal{P}$.

Definition 1.1. $A \subseteq 2^{<\omega}$ is called *low for speed* iff it has the following property: Whenever R is a recursive set and Ψ is a recursive functional such that $\Psi^A = R$, there is a total recursive functional Θ and a polynomial p such that for all $\sigma \in 2^{<\omega}$,

$$\Theta(\sigma) \downarrow = R(\sigma) \ \ \& \ \ t(\Theta, \sigma) \leq p(t(\Psi^A, \sigma))$$

That is, knowing A provides no more than a polynomial time advantage when computing recursive sets. We will write **LFS** for the set of all $A \subseteq 2^{<\omega}$ which are low for speed.

The notion of low for speed is only interesting if we restrict ourselves to non-recursive sets, as shown by the following theorem.

Theorem 1.4. *Let R be a recursive set. Then $R \in$ **LFS** $\Leftrightarrow R \in \mathcal{P}$.*

Proof. ($\Rightarrow$) Suppose R is low for speed, and let Ψ be the identity oracle machine which simply copies its input to the query tape and then accepts or rejects based on whether the input is in the oracle. Then $\Psi^R = R$ and $t(\Psi^R, \sigma) \leq C|\sigma|$ for some constant C. As $R \in$ **LFS**, there is some polynomial p and some total recursive Θ such that $\Theta = R$ and $t(\Theta, \sigma) \leq p(t(\Psi^R, \sigma)) \leq p(C|\sigma|)$. Therefore, $R \in \mathcal{P}$.

($\Leftarrow$) Suppose $R \in \mathcal{P}$, and let Ψ be a functional such that $\Psi^R = S$ is some recursive set. We must construct a functional Θ and a polynomial p such that $\Theta = S$ and $t(\Theta, \sigma) \leq p(t(\Psi^R, \sigma))$.

Let Φ be a functional witnessing $R \in \mathcal{P}$. That is, $\Phi = R$ and there is some polynomial q such that $t(\Phi, \sigma) \leq q(|\sigma|)$.

We will define $\Theta(\sigma)$ as follows: on input σ, run $\Psi^R(\sigma)$ but whenever a query state is reached, run Φ on the value printed on the query tape and act accordingly. Let $t = t(\Psi^R, \sigma)$. Then every query must have length no more than t and so each action of Θ corresponding to one step of Ψ^R takes no more than poly(q(t)) steps to run Φ and act appropriately. By replacing q with a larger polynomial, we may assume each simulation takes no more than $q(t)$ time. Therefore, we have:

$$\begin{aligned} t(\Theta, \sigma) &\leq \sum_{i=1}^{t} (\text{number of steps needed to run } \Phi \text{ on query from stage i}) \\ &\leq \sum_{i=1}^{t} q(t) \\ &= t \cdot q(t) \end{aligned}$$

And so $R \in \mathbf{LFS}$

□

Note that theorem 1.4 is only true using the query machine model of oracle computation, and not for the characteristic function tape model. In particular, while the $\Leftarrow$ direction remains true, the proof for the $\Rightarrow$ direction depends critically on the query tape model.

We can also relativize our definition of low for speed as follows, though this will not be used until chapter 4.

Definition 1.2. $A \subseteq 2^{<\omega}$ is called *low for speed relative to* B iff it has the following property: Whenever $R \leq_T B$ and Ψ is a recursive functional such that $\Psi^{A \oplus B} = R$, there is a total recursive functional Θ and a polynomial p such that for all $\sigma \in 2^{<\omega}$,

$$t(\Theta^B, \sigma) \leq p(t(\Psi^{A \oplus B}, \sigma))$$

Unlike most other properties that are given the name *low*, it is easy to see that **LFS** is not closed under relative computation. In fact, we have

Theorem 1.5. *Every m-degree contains a set which is not low for speed.*

Proof. Let $A \subseteq 2^{<\omega}$ be given, and let B be a recursive set not in $\mathcal{P}$. Note that such a set exists by the time hierarchy theorem [7]. Then $A \oplus B \equiv_m A$, but $A \oplus B$ is not low for speed since B can be computed from $A \oplus B$ in time $O(n)$ but $B \notin \mathcal{P}$. □

We can, however, get downward closure if we restrict ourselves to polynomial-time reductions.

Definition 1.3. If $A, B \subseteq 2^{<\omega}$, we write $A \leq_T^P B$ and say A *is polynomial-time reducible to* B if there is a total recursive functional Φ and a polynomial p such that $\Phi^B = A$ and $t(\Phi^B, \sigma) \leq p(|\sigma|)$. If $A \leq_T^P B$ and $B \leq_T^P$, we write $A \equiv_T^P B$.

We denote by $\mathcal{P}^A$ the set $\{B : B \leq_T^P A\}$.

Theorem 1.6. *If $B \leq_T^P A$ and $A \in$ **LFS**, then $B \in$ **LFS**.*

Proof. The proof is very similar to the proof of the $\Leftarrow$ direction of theorem 1.4. In particular, given some $\Psi^B = R$ with R recursive, we can construct a machine Γ such that $\Gamma^A = \Psi^B$ by replacing any queries to B with A-computations of B. Because $B \leq_T^P A$, we can do this in a way such that there is some polynomial p such that

$$\forall \sigma \;\; t(\Gamma^A, \sigma) \leq p(t(\Psi^B, \sigma))$$

Then since $A \in$ **LFS**, there is some functional Θ and some polynomial q such that $\Theta = \Gamma^A = \Psi^B = R$ and

$$\forall \sigma \;\; t(\Theta, \sigma) \leq q(t(\Gamma^A, \sigma)) \leq q(p(t(\Psi^B, \sigma)))$$

and so B is low for speed. □

Chapter 2

Existence

In this chapter, we will construct an r.e. set which is low for speed using a priority argument.

2.1 Some very non-low sets

In chapter 1, we gave examples of some sets which are not low for speed and noted that such sets are extremely common. Before constructing our low for speed set, we give examples of sets which not only fail to be low for speed, but are as far from it as possible.

We start with an easy example. Let $\{W_e\}$ be a recursive enumeration of all r.e. sets. Then the set $A = \{\langle e, \sigma\rangle : \sigma \in W_e\}$ is r.e. and for every r.e. W we have $W \leq_T^P A$. In fact, this reduction is not just polynomial-time, but is in fact linear since we may choose our pairing function such that $|\langle e, \sigma\rangle| \leq |e| + |\sigma| + c$ for some constant c.

Sets such as the one above could reasonably be called *high for speed*. It is then natural to ask where in the structure of the r.e. degrees one can find such sets, and therefore what properties they can have. For our next example, we will need the following definition.

Definition 2.1. Two r.e. sets A, B are said to form a *minimal pair* iff whenever $C \leq_T A$ and $C \leq_T B$, $C \leq_T 0$.

The construction of a minimal pair is due independently to Lachlan [10] and Yates [20]. By adapting Lachlan's proof showing the existence of a minimal pair of high sets, we can get the following theorem. The proof of this theorem also provides a relatively straightforward example for our first $0''$-priority construction.

Theorem 2.1. *There is a minimal pair of r.e. sets A, B such that whenever R is recursive $R \leq_T^P A$ and $R \leq_T^P B$.*

Proof. We will build A, B as subsets of $2^{<\omega}$ using a $0''$-priority construction.

Requirements

Let $\{\Phi_e\}_{e\in\omega}$ be a recursive listing of all functionals. We will ensure that $\Phi_e \leq_T^P A$ and $\Phi_e \leq_T^P B$ for all total Φ_e by building A, B such that $A^{[e]} =^* \Phi_e$ and $B^{[e]} =^* \Phi_e$ when Φ_e is

total. Here we are considering $\langle \cdot, \cdot \rangle$ to be a function from $\mathbb{N} \times 2^{<\omega} \to 2^{<\omega}$. We therefore must meet the following requirements:

- N_e : If $\Phi_e^A = \Phi_e^B = R$, then R is recursive.
- P_e^A : $\exists \sigma \ (\Phi_e(\langle e, \sigma \rangle) \neq A(\langle e, \sigma \rangle))$.
- C_e^A : If Φ_e is total, then $A^{[e]} =^* \Phi_e$.
- P_e^B, C_e^B : As above, but with B instead of A.

Remark. Here we are using Posner's trick as described in IX.1.4 of [18]. Namely, instead of considering pairs Ψ, Γ such that $\Psi^A = \Gamma^B$, it is sufficient to consider only a single Φ since the requirements ensure $A \neq B$ and thus if $\sigma_0 \in A - B$, we can build a single Φ such that $\Phi^C(\sigma) = \Psi^C(\sigma)$ if $\sigma_0 \in C$ and $\Phi^C(\sigma) = \Gamma^C(\sigma)$ otherwise.

Strategies

We now describe how a single strategy works.

Strategy for P_e

We will meet the P_e^A requirements by the standard diagonalization method. Namely, our strategy will be as follows:

1. Pick some $\sigma \in 2^{<\omega}$ with $\langle e, \sigma \rangle$ having length larger than anything referenced so far in the construction and keep $\langle e, \sigma \rangle$ out of A.
2. Wait until $\Theta_e(\langle e, \sigma \rangle) \downarrow = 0$.
3. Enumerate $\langle e, \sigma \rangle$ into A and stop.

There are two possible outcomes for P_e^A:

- Wait in step (2) forever. Then $\Theta_e(\langle e, \sigma \rangle) \uparrow \neq A(\langle e, \sigma \rangle)$.
- Stop after reaching step (3). Then $\Theta_e(\langle e, \sigma \rangle) = 0$ but $A(\langle e, \sigma \rangle) = 1$.

In either case, the strategy ensures that requirement P_e^A is met.

A P_e^A strategy is said to be *injured* if the $\langle e, \sigma \rangle$ it is monitoring is put into A without its consent. In this case, the strategy must pick a new, longer σ and start over.

Strategies for P_e^B are defined in the obvious way.

Strategy for C_e^A

1. Pick some N_0 larger than the length of anything mentioned in the construction so far. Require all computations to assume $A^{[e]} = \Phi_e$ for strings of length exceeding N_0. Start with σ as the first string in $2^{<\omega}$ of length N_0 (assume we have ordered $2^{<\omega}$ in some way so that shorter strings appear before longer ones).

2. Wait until $\Phi_e(\sigma)\downarrow$.

3. If it gives 1, put σ into $A^{[e]}$. If it gives 0, keep σ out of $A^{[e]}$. In either case, pick the next σ in our ordering of $2^{<\omega}$ and go back to step (2).

There are two possible outcomes here. We can either choose infinitely many σ because Φ_e is total, or we will find a σ such that $\Phi_e(\sigma)\uparrow$. We will call the former outcome ∞ and the latter w (for "w"ait). Note that ∞ is a Π^0_2 outcome and w is Σ^0_2.

A C_e strategy is injured if some string of length greater than N_0 enters $A^{[e]}$ and we haven't yet seen $\Phi_e(\sigma)\downarrow = 1$. In that case, it must pick a new N_0 and start over.

Strategy for N_e

We can ensure that $\Phi^A_e = \Phi^B_e = R \to R$ is recursive by doing the following:

1. Start with $n = 0$.

2. Wait until we see $\Phi^{A_s}_e \upharpoonright n = \Phi^{B_s}_e \upharpoonright n$.

3. When we do, select either A or B to restrain. Suppose it's A. Do not allow any strings of length less than or equal to $\max\{\mu(\Phi^{A_s}_e, \sigma) : |\sigma| \leq n\}$ to enter A. (If it's B, use a similar restraint on B).

4. Increment n by one and go back to step 2.

As with the C_e strategies, there are two possible outcomes to this strategy. We can either increment n forever because we see increasingly longer lengths of agreement, or we eventually end in step 2 waiting forever. As before, we will call the former outcome ∞ and the latter w (for "w"ait) and note that ∞ is a Π^0_2 outcome and w is Σ^0_2.

Remark. The naming conventions used for requirements is largely historical. The P_e requirements are so named because they can be satisfied by taking the positive action of putting certain strings into our sets. The C_e requirements deal with coding. The N_e requirements are said to be negative requirements because they can be met by keeping strings out of our sets.

Tree of strategies and the construction

We begin by ordering the requirements in some effective way such that for all e, P^A_e comes before C^A_e in the list and likewise for P^B_e and C^B_e. Let o be the only outcome of the P strategies and let $\Lambda = \{o, w, \infty\}$ be the set of all outcomes and order it by $o <_\Lambda \infty <_\Lambda w$. Our tree of strategies will be a tree $T \subseteq \Lambda^{<\omega}$. We want to interpret level i of our tree as corresponding with strategies for the i^{th} requirement in our list and if $\gamma \in T$, then $\gamma(i)$ corresponds to the outcome of strategy i. Formally, we say

$$T = \{\gamma \in \Lambda^{<\omega} : \gamma(i) = o \leftrightarrow \text{requirement } i \text{ is a } P \text{ requirement}\} \tag{2.1}$$

By convention, we think of strategy trees as growing downward. We can then picture this tree as in figure 2.1. At each stage of the construction, we will traverse down through the

tree following the path that currently appears to be correct and let each strategy along that path act. When a strategy acts, it assumes that it is correct about the eventual outcome of requirements above it in the tree. For example, the circled strategy in figure 2.1 takes its actions assuming that N_0 will no longer see agreement between Φ_0^A and Φ_0^B, and that Φ_0 is total and so $A^{[0]}$ will be equal to Φ_0 above the N_0 picked by C_0^A.

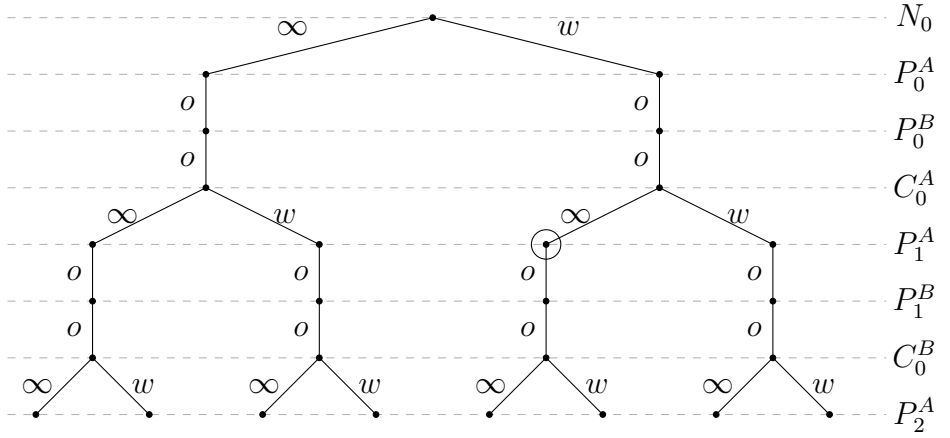

Figure 2.1: An example of a tree of strategies used in the proof of theorem 2.1. The circled strategy acts under the assumption that N_0 will have outcome w and C_0^A will have outcome ∞.

Let γ, δ be strategies on T. We say γ has higher priority than δ iff either γ is above δ ($\gamma \prec \delta$) or γ is to the left of delta ($\gamma(i) <_\Lambda \delta(i)$ for the least i where they differ). Throughout the construction, we will only require that strategies respect restraints imposed by higher priority strategies and may freely ignore restraints imposed by lower priority strategies. That is, we allow higher priority strategies to injure lower priority ones.

Because we are building an r.e. set, the only real action we can take is putting strings into A or B. Both P and C strategies need to do this in order to satisfy their requirements, so we describe the procedure for doing so. Let γ be some strategy on T and suppose it wants to put some σ into A (putting σ into B is similar). If σ has length less than an A-restraint imposed by a strategy to the left of γ, then it cannot do it and must give up on putting n into A (in practice, this will not happen since any restraints imposed by strategies to the left of γ will cause γ to be injured and pick an n above that restraint). γ can ignore any restraints imposed by strategies to the right of it since they all have lower priority. Therefore, we need only consider restraints imposed by strategies above γ on the tree. Any restraints imposed by P or C strategies must be respected, as must any restraints imposed by N strategies that γ thinks will have outcome w (again, this will not happen since such restraints would have injured γ when they were put in place). However, if γ believes that the N_e strategy will have outcome ∞, then it can wait to see an increase in length of agreement and then ask N_e to drop the A-restraint entirely and put σ into A at that point. We will need to do some

bookkeeping work to ensure that the restraints drop back simultaneously and that we drop back both A and B infinitely often, but this is the general idea.

We must also ensure that the computations used by N strategies are not injured by coding strategies that may try to put infinitely many strings into $A^{[e]}$ arbitrarily late. We will say a computation $\Psi^{A_s}(\sigma)$ is C_e^A-compatible above M at stage s if whenever the computation queries whether $\langle e, \tau\rangle \in A_s$ with $|\tau| \geq M$, we have $\Phi_e(\tau) \downarrow = A_s(\langle e, \tau\rangle)$ in $\leq s$ steps. We define C_e^B-compatibility in the same way.

We are now ready to describe the construction. At stage s, we will construct a path γ of length s through T. We will build it in stages, and start with $\gamma_0 = \lambda$, the empty string. We start with two initially empty sets Q_A, Q_B that will hold the strings the strategies want to put into A or B, respectively. Suppose γ_i has been defined. This represents a strategy in T so let it act as follows based on what type of strategy it is:

(γ_i is a P_e^A strategy): Follow the strategy as described previously. If $\Phi_e(\langle e, \sigma\rangle) \downarrow = 0$ in $\leq s$ steps and $\sigma \notin A_s$, then try to put σ into A. If restraints from ∞-outcome N strategies above γ prevent this, put σ into Q_A instead. Set $\gamma_{i+1} = \gamma_i \hat{} \, o$.

(γ_i is a C_e^A strategy): If we do not currently have an N_0, pick one larger than the length of anything mentioned so far in the construction and impose the restraint that no lower priority strategy can put $\langle e, \sigma\rangle$ into A for any σ with $|\sigma| > N_0$. Either way, now check if $\Phi_e(\sigma) \downarrow$ in $\leq s$ steps for the first σ we have not yet seen Φ_e converge on. If it does not converge, set $\gamma_{i+1} = \gamma_i \hat{} \, w$ and continue down the tree.

If it does converge and $|\sigma| > N_0$, then attempt to put σ into $A^{[e]}$. If restraints from ∞-outcome N strategies above γ prevent this, put $\langle e, \sigma\rangle$ into Q_A instead. Choose the next σ to monitor and set $\gamma_{i+1} = \gamma_i \hat{} \, \infty$.

(γ_i is an N_e strategy): Find the maximal n such that $\Phi_e^{A_s} \upharpoonright n = \Phi_e^{B_s} \upharpoonright n$ and such that all the computations involved are compatible (as described earlier) with all C strategies that γ_i believes have outcome ∞. If this n has increased since last time this strategy was visited, set $\gamma_{i+1} = \gamma_i \hat{} \, \infty$. If it has not increased, set $\gamma_{i+1} = \gamma_i \hat{} \, w$.

The C_e^B and P_e^B strategies are handled in the obvious way.

Suppose we have now built our path γ. We must now pick whether to drop A or B restraints for each N strategy that found an increased length of agreement. If $\min\{|\sigma| : \sigma \in Q_A\} \leq \min\{|\sigma| : \sigma \in Q_B\}$, then have all such strategies drop their A restraint and put everything from Q_A into A. Each such N strategy now imposes a B-restraint equal to the maximum use from any B-computation it used when finding the increased length of agreement. If $\min Q_B < \min Q_A$, do the same with A and B reversed. Injure all strategies to the right of γ. This concludes stage s.

Verification

We must now verify that our construction actually satisfies all the requirements. Let $f \in [T]$ be the leftmost path visited infinitely often during the construction. Formally, if γ_s is the path built at stage s, then $f = \liminf_s \gamma_s$.

Remark. Note that $f \leq_T 0''$. This is the reason constructions like this are called $0''$-priority arguments.

Lemma 2.2. *f is correct about the outcome of all strategies on it.*

Proof. We go by induction on i. Start with $i = 0$. Now assume $f(j)$ is correct for $j < i$. We must show $f(i)$ is correct. Let $\gamma_i = f \upharpoonright i$. If γ_i is a P strategy, then we trivial get $f(i)$ being correct since there is only one possible outcome. If γ_i is an N strategy, then $f(i) = \infty$ iff we see $\Psi^A \upharpoonright n = \Psi^B \upharpoonright n$ for infinitely many n and so $\Psi^A = \Psi^B$. If γ_i is a C strategy, then $f(i) = \infty$ iff $\Phi_e(\sigma) \downarrow$ for all σ and so Φ_e is in fact total. □

Lemma 2.3. *Any strategy on the true path is injured only finitely many times.*

Proof. By the definition of f, there is some stage s such that the construction never goes left of f after stage s. Therefore, only finitely many strategies to the left of f are ever visited and each is visited only finitely many times, so they can collectively cause only a finite number of injuries to f. All strategies to the right of f have lower priority than every strategy on f and by our construction respect all restraints imposed by higher priority strategies. Therefore, we need only be concerned with some $\gamma_i = f \upharpoonright i$ being injured by strategies above it. We go by induction on i to show γ_i is injured only finitely many times. γ_0 is the root of the tree and has higher priority than anything else and is therefore never injured. Suppose γ_j is injured only finitely many times for all $j < i$ and consider γ_i.

(γ_i is a P_e strategy): P_e^A strategies are only injured if the $\langle e, \sigma \rangle$ they are monitoring enters A before they see $\Phi_e(\langle e, \sigma \rangle) \downarrow = 0$. The only other strategy that might do this is the strategy for C_e^A, but by our ordering of the requirements we ensured that such a strategy must appear lower in the tree. P_e^B strategies are similar. Therefore, a P_e strategy is never injured by a strategy above it in the tree.

(γ_i is a C_e strategy): C_e^A could only possibly be injured by P_e^A since they are the only ones dealing with $A^{[e]}$. Since P_e^A acts at most once after each time it is injured and we know P_e^A is injured at most finitely often, we conclude C_e^A is injured at most finitely often.

(γ_i is an N_e strategy): Such a strategy is injured iff a higher priority strategy puts a short string into A or B despite γ_i's restraint. We have already seen that $P_{e'}$ strategies on f act only finitely many times, so the total number of injuries due to $P_{e'}$ strategies above γ_i must be finite. Suppose $j < i$ and γ_j is a $C_{e'}$ strategy. If $f(j) = w$, then by lemma 2.2, γ_j eventually stops seeing $\Phi_{e'}$ converging and so stops putting in strings. If $f(j) = \infty$, then γ_i only ever considers computations that are consistent with all the eventual actions of γ_j. Therefore, γ_i is injured at most finitely many times.

□

Lemma 2.4. *Every strategy on the true path successfully satisfies its requirement.*

Proof. Let γ be a strategy on the true path, and suppose it is not injured after stage s_0.

If γ is a P or C strategy, it is enough to show that when it requests to put strings into either A or B it is eventually successful. Suppose it wants to put σ into A at stage $s > s_0$. If it doesn't do it immediately, then it must be because it put it into Q_A and Q_B was selected at the end of stage s. Since γ is not injured after stage s and γ is visited infinitely often, it will continue to put σ into Q_A until it is successful at getting σ into A. Since there are

only finitely many τ with $|\tau| \leq |\sigma|$, eventually Q_B will not contain any strings shorter than σ and therefore σ will go into A at that time (if not before).

Now suppose γ is a strategy for N_e. If $\Phi_e^A \neq \Phi_e^B$, then the requirement is satisfied trivially. Suppose now that $\Phi_e^A = \Phi_e^B$. Let s_0 be the last stage at which N_e is injured. We will now build a recursive Θ such that $\Theta = \Phi_e^A = \Phi_e^B$ as follows. On input σ, monitor the construction of A, B until a stage $s > s_0$ where γ is visited and $\Phi_e^{A_s} \upharpoonright m = \Phi_e^{B_s} \upharpoonright m$ for some $m \geq |\sigma|$ and all computations are compatible with any C_e strategies above γ that γ believes have outcome ∞. When such a stage is found, output $\Theta(\sigma) = \Phi_e^A(\sigma) = i$ and halt.

Since γ is on the true path, it will be visited infinitely often and since $\Phi_e^A = \Phi_e^B$ we will eventually find an m such that $\Phi_e^A \upharpoonright m = \Phi_e^B \upharpoonright m$. By our previous lemma, γ will have the correct guess about the outcome of any C strategies above it, so we can in fact wait for C-compatible computations. Therefore, Θ is total. Since N_e will never be injured, the restraints imposed on A or B guarantee that either $\Phi_e^{A_{s'}}(\sigma) = i$ or $\Phi_e^{B_{s'}}(\sigma) = i$ for all stages $s' \geq s$. Therefore, Θ is correct and total as needed.

□

As f contains a strategy for every requirement, A is as required.

□

2.2 Existence of an r.e. set which is low for speed

Theorem 2.5. *There is a non-recursive r.e. set which is low for speed.*

We will use a $0''$-priority argument to construct an r.e. set A satisfying the theorem.

To simplify our notation, we will think of A as a subset of $\mathbb{N}$. We will build A in stages and will use A_s to denote the approximation to A at stage s. When $\alpha \in 2^{<\omega}$, we will say "α is an initial segment of A" or "A extends α", and write $\alpha \prec A$ iff $A \upharpoonright |\alpha| = \alpha$. We will say α is "A-accessible at stage s" iff $A_s \upharpoonright |\alpha| \subseteq \alpha$. Intuitively, a string α is A_s-accessible if we can make α an initial segment of A by enumerating more things into A.

Remark. For the reader who wishes to build A as a subset of $2^{<\omega}$ instead of a subset of $\mathbb{N}$, we can consider building A as a subset of $\{1^n : n \in \mathbb{N}\}$. Then everywhere in the proof where we say $n \in A$, interpret it as $1^n \in A$, and so on.

The requirements

In order to accomplish our goal, we must meet the following requirements:

- For each $e \in \omega$, P_e: $A \neq \Theta_e$ where $\{\Theta_e\}_{e \in \omega}$ is some effective ordering of all partial recursive functionals.
- For all functionals Ψ and R, $L_{\Psi,R}$: If $\Psi^A = R$ and R is total, there is some total Φ such that $\Phi = R$ and some polynomial p such that $t(\Phi, \sigma) \leq t(\Psi^A, \sigma)$.

The P_e requirements ensure that A is non-recursive, and the $L_{\Psi,R}$ requirements ensure it is low for speed.

The strategies

Strategy for P_e

We will meet the P_e requirements in the usual way. Namely, our strategy will be as follows:

1. Pick some $n \in \mathbb{N}$ with $|n|$ larger than anything referenced so far in the construction and keep n out of A.
2. Wait until $\Theta_e(n) \downarrow = 0$.
3. Enumerate n into A and stop.

There are two possible outcomes for P_e:

- Wait in step (2) forever. Then $\Theta_e(n) \uparrow \neq A(n)$.
- Stop after reaching step (3). Then $\Theta_e(n) = 0$ but $A(n) = 1$.

In either case, the strategy ensures that requirement P_e is met.

A P_e strategy is injured if its n is put into A (presumably by a higher priority strategy) before it sees $\Theta_e(n) \downarrow = 0$. If this happens, the strategy must forget about the n it was monitoring and start over with a new one the next time it is visited.

Strategy for $L_{\Psi,R}$

Our goal is to enumerate a recursive Φ that is no worse than polynomial-time slower than Ψ^A. To accomplish this, we will monitor $\Psi^\alpha(\sigma)$ for various α that are A-accessible at stage s. Whenever we see $\Psi^\alpha(\sigma) \downarrow = i$ taking s steps, we will enumerate $\Phi(\sigma) = i$ at stage s. In this case, we will say Φ *simulates* Ψ.

In order to succeed in meeting our requirement, we must ensure that this simulation takes no more than poly(s) many steps and therefore we cannot monitor more than poly(s) α at stage s. However, there are generally exponentially many α that are A accessible at stage s with $|\alpha| = s$. Therefore, we restrict ourselves to monitoring only linearly many possible α. At stage s, for any $k \leq s$ with $k \notin A_s$, we define $\alpha_k = (A_s \upharpoonright k) \cup [k, s]$, and only monitor computations using α of this form. That is, we assume that if we put k into A at stage s, then all numbers between k and s will also go into A. It can be helpful to visualize the α_k as a collection of threads branching off from our current guess of A. This is visualized in figure 2.2.

Our construction will maintain a list of branch points to monitor. In practice, we will monitor those α_k where k is a number that one of the P_e strategies is monitoring and therefore could eventually want to put into A; this will be described in more detail later. At stage s, the strategy for $L_{\Psi,R}$ will run $\Psi^{\alpha_k}(\sigma)$ for s steps for each k being monitored and also run $\Psi^{A_s}(\sigma)$ for s steps. If we see $\Psi^{\alpha_k}(\sigma) \downarrow = i$, enumerate $\Phi(\sigma) = i$ if $\Phi(\sigma)$ has not already been

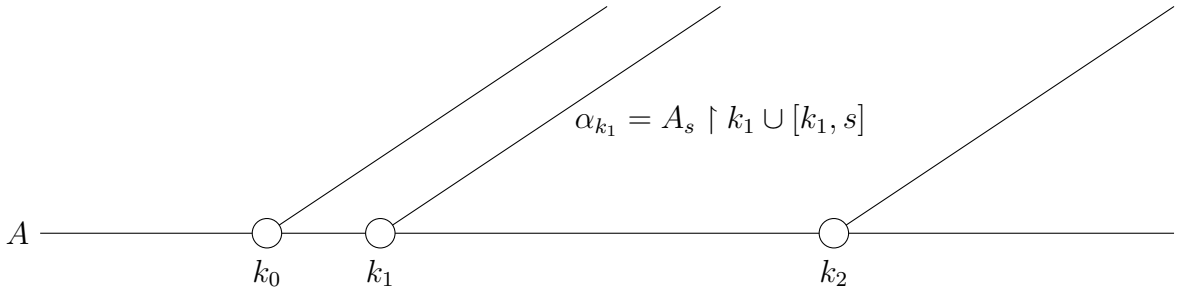

Figure 2.2: A visual representation of the α_k along which we monitor Ψ. At stage s we will check if $\Psi^{\alpha_k}(\sigma) \downarrow$ in $\leq s$ steps and simulate if so.

defined. We will eventually show that this can be done in a way so that there is a polynomial p such that $t(\Phi, \sigma) \leq p(t(\Psi^A, \sigma))$.

Presuming such a polynomial exists, we must now ensure two things.

First, we must make sure that $\Phi(\sigma)$ is fast enough. Because we have been monitoring computations of increasing lengths, the fact that we had not yet defined $\Phi(\sigma)$ must mean that there was no computation using any of the currently monitored α_k that took less than s steps. Therefore, we need only be concerned with computations along future α that are not currently being monitored. We thus require that any future k added to the monitored set have the property that $|k| > s$, and so we know that computations along such a thread that we didn't see as part of $\Psi^{A_s}(\sigma)$ must have use greater than s and therefore take more than s steps.

Second, we must make sure that our simulator, $\Phi(\sigma)$, actually gives the correct answer. To do this, we must ensure that it remains possible that $\alpha_k \prec A$ which we can do by forbidding any $n < k$ from entering A. However, we cannot impose arbitrarily long restrictions for each σ separately forever and have any hope of allowing other strategies to act. Therefore, we need a way to eventually drop this restraint. Because we only need Φ to be total and correct if R is total with $\Psi^A = R$, we can wait for $R(\tau) \downarrow$ for all τ with $|\tau| \leq |\sigma|$. While we could wait just for $R(\sigma) \downarrow$, this would cause problems for our simulators being able to follow the construction since they cannot search over all possible strings in polynomial time. If we see $R(\sigma) \downarrow = i$, we say R *protects* the computation since we are then guaranteed that $\Phi(\sigma)$ is correct. If $R(\sigma) \downarrow \neq i$, then we force $R \neq \Psi^A$ by ensuring $\alpha_k \prec A$ and then stop worrying about this requirement. If $R(\sigma) \uparrow$, then we need not worry further about this (Ψ, R) pair.

An $L_{\Psi,R}$ strategy therefore has three outcomes. We name these three outcomes as follows:

1. ∞: $\forall \sigma \; R(\sigma) \downarrow = \Psi^A(\sigma)$
2. s_1: $\exists \sigma \; R(\sigma) \downarrow \neq \Psi^A(\sigma)$
3. s_2: $\exists \sigma \; R(\sigma) \uparrow$

Note that ∞ is a Π^0_2 outcome, s_1 is Σ^0_1, and s_2 is Σ^0_2.

An $L_{\Psi,R}$ strategy is injured if either:

1. It has seen $\Psi^{\alpha_k} \downarrow = i$ but has not yet seen $R(\sigma) \downarrow = i$ and some number less than k enters A.
2. It has seen $\Psi^{\alpha_k} \downarrow = i$, has seen $R(\sigma) \downarrow \neq i$, and has put α_k into A and then some number less than k enters A.

In either case, an injury causes the strategy to forget all computations it had seen or simulated so far and begin creating a new simulator. This means also that any diagonalization it had accomplished while trying to force outcome s_1 is erased as well.

Tree of strategies and the construction

Order the requirements in some effective way with order type ω and order the outcomes for L requirements by $s_1 <_L \infty <_L s_2$. We will define our tree level-by-level. The i^{th} level will consist of strategies for the i^{th} requirement. At an $L_{\Psi,R}$ node, we will branch three times and label the branches ∞, s_1, s_2 corresponding to the possible outcomes. No branching is necessary at nodes corresponding to P_e strategies (if we want to be precise, we can say there is one branch labeled by some new outcome symbol). We identify a strategy with the string representing its position in the tree. An example tree is shown in figure 2.3.

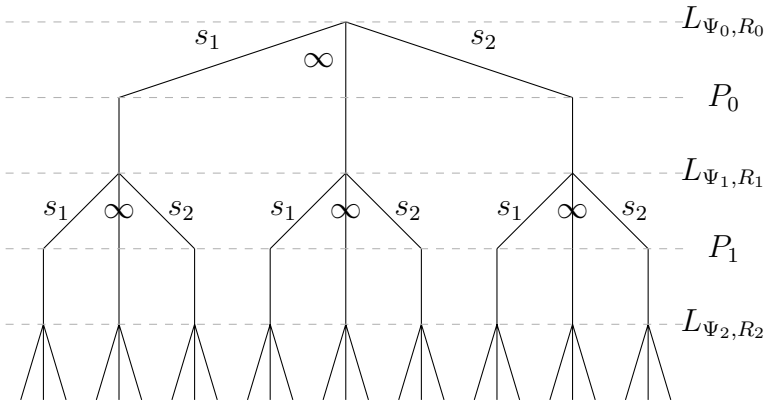

Figure 2.3: An example tree of strategies.

We say a strategy γ has higher priority than any strategy δ which is either below it or to the right of it in the tree. That is, if either $\gamma \prec \delta$ or if $\gamma(i) <_L \delta(i)$ for the least i such that $\gamma(i) \neq \delta(i)$.

We will now describe how strategies act based on their position in the tree and the assumptions they have about higher-priority strategies. Because we are building an r.e. set, the only action that can be taken is putting numbers into A.

Suppose some strategy γ wants to put n into A at stage s. We must get permission from all higher-priority strategies to do so. We proceed back up the tree to ask each $L_{\Psi,R}$ strategy for permission in turn, and act based on γ's guess about the eventual outcome of that strategy:

- s_1: γ is working under the assumption that $\Psi^A \neq R$. Since $L_{\Psi,R}$ has higher priority than γ, the last injury to γ must have occurred after the diagonalization and so $|n|$ must be larger than anything used to accomplish that diagonalization. As this is the only restraint imposed by a $L_{\Psi,R}$ that has successfully been diagonalized against, γ may safely put n into A.

- s_2: We are assuming R is partial and so need not respect any restraints imposed by $L_{\Psi,R}$. Therefore, γ may safely put n into A.

- ∞: We are assuming R is total and we have not diagonalized against Ψ so we need to respect any restraints in place. $L_{\Psi,R}$ will prevent γ from putting n into A only if it would make an unprotected computation cease to be A-accessible. That is, if there is some σ and some $m > n$ such that $\Psi^{\alpha_m}(\sigma)\downarrow = i$ has been simulated, but we have not yet seen $R(\tau)\downarrow = i$ for all τ with $|\tau| < |\sigma|$. This setup is shown visually in figure 2.4.

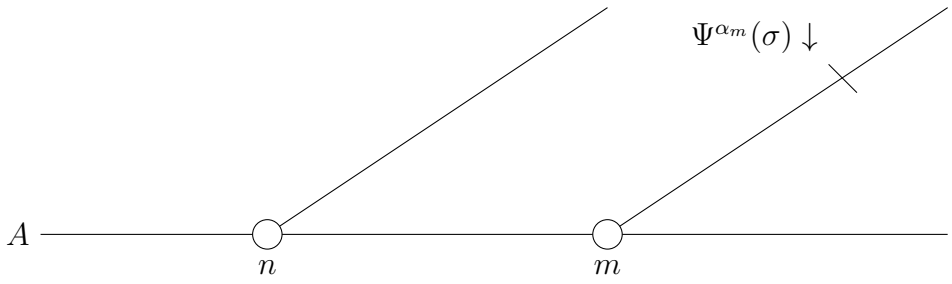

Figure 2.4: We cannot put n into A until $R(\sigma)\downarrow$ to protect the simulated value.

Because γ assumes that R is total, it can wait for R to converge and then put n into A. Therefore, any α_m with $m > n$ will not be initial segments of A and so with γ's priority we remove them from the list of monitored threads. In order to allow the simulator that $L_{\Psi,R}$ is building to keep track of the active threads in polynomial time, we need to wait to give the simulator time to "see" these actions taking place. By the recursion theorem, we can find s' such that it takes the simulator s' steps to see that this request was made. γ now imposes the global restraint (with priority γ) that we only simulate along α_k for $k \leq n$ until a stage t such that we can see that $R(\tau)$ is defined for all τ with $|\tau| < s'$ and can see so in t steps. All future visits to γ or an $L_{\Psi,R}$ strategy of lower priority must check if this condition has been met. Suppose we eventually find such a t. Then we now have permission from $L_{\Psi,R}$ to put n into A. Continue to maintain the restraint on $L_{\Psi,R}$'s monitored set and proceed up the tree to the next $L_{\Psi,R}$ strategy.

Once we have visited every higher priority strategy and gotten permission from all of them, put α_n into A and drop any restraints γ imposed on monitoring.

We can think of the above procedure as a way to ask any $L_{\Psi,R}$ of higher priority than the strategy that wants to put n into A to move its monitoring to threads α_k with $k \leq n$. This then allows us to free put n into A since α_n must have been a monitored computation.

The construction

Our construction will be in stages by recursion on s.

Stage $s = 0$: Set $A_0 = \emptyset$ and do nothing.

Stage $s + 1$: Traverse the tree of strategies to depth s, defining a length s path δ by recursion. $\delta(0)$ is just the root of our tree. Suppose we have defined our path up length $i < s$. Our action depends on whether $\gamma = \delta \upharpoonright i$ is a P_e strategy or an $L_{\Psi,R}$ strategy.

If γ represents a strategy for P_e, do the following:

1. If this strategy does not currently have an $n \in \mathbb{N}$ it is monitoring, pick one with $|n|$ larger than anything mentioned so far in the construction. Add α_n to the list of things to be monitored for all $L_{\Psi,R}$ strategies that have not had their monitored set restricted by a strategy of priority higher than γ and injure any strategies of lower priority. Permanently prohibit any strategies of lower priority from putting n into A. If we already have an n from a previous visit to γ, proceed directly to step 2.

2. Run s steps of $\Theta_e(n)$. If it does not converge or if $\Theta_e(n) = 1$, do nothing at this stage.

3. If $\Theta_e(n) \downarrow = 0$ in $\leq s$ steps, put α_n into A using the previously described method.

Set $\delta(i)$ to be the only successor of γ.

If γ represents a strategy for $L_{\Psi,R}$, do the following:

If we have previously reached the s_1 outcome for this strategy and have not been injured since, then do nothing. Otherwise, run s steps of $\Psi^{\alpha_k}(\sigma)$ for all α_k currently in the monitored set and for all σ with $|\sigma| < s$. For each σ, simulate the shortest computation that we see converge by enumerating $\Phi(\sigma) = i$. Look for α_k, σ such that the following three properties hold:

- $\Psi^{\alpha_k}(\sigma) \downarrow \neq R(\sigma) \downarrow$
- $R(\tau) \downarrow$ for all τ with $|\tau| \leq |\sigma|$ and the total time to see this is $\leq s$.
- k is larger than any restrictions placed by higher priority P_e strategies or higher priority $L_{\Psi,R}$ strategies that γ believes have outcome s_1.

If we can find such a k, σ, use the previously described method to put α_k into A. Set our outcome to be s_1, injure all lower priority strategies, and permanently prohibit any lower priority strategies from putting a number less than k into A.

Set $\delta(i)$ according to our current guess as to the outcome of this strategy. Namely, if γ has successfully diagonalized to ensure outcome s_1, either at this stage or a previous one, set $\delta(i) = s_1$. Otherwise, find the maximum n such that $R(\sigma) \downarrow$ in $\leq s$ steps for all σ with

$|\sigma| \leq n$. If this value has increased since we last visited γ, set $\gamma(i) = \infty$ and injure all strategies below the s_2 outcome. Otherwise, set $\delta(i) = s_2$.

Verification

We must now verify that our construction actually satisfies the requirements. As usual we define the true path, f, to be the leftmost path visited infinitely often. That is, $f = \liminf_s \delta_s$ where δ_s is the path we built at stage s.

Lemma 2.6. *The true path is correct about the outcome of each strategy.*

Proof. We go by induction on $f \upharpoonright i$. There is only one possible outcome for P_e strategies, so we need only consider $L_{\Psi,R}$ strategies.

Suppose $\gamma = f \upharpoonright i$ is a strategy for $L_{\Psi,R}$. Suppose stage s is the last stage at which γ is injured.

If $f(i) = s_1$, it must be because we found an α_k, σ after stage s such that $\Psi^{\alpha_k}(\sigma) \downarrow \neq R(\sigma) \downarrow$. Because γ was not injured after this, our restraint on A was never violated and therefore $\alpha_k \prec A$ and so $\Psi^A(\sigma) = \Psi^{\alpha_k}(\sigma) \neq R(\sigma)$.

If $f(i) = s_2$, it must be because we at some point failed to find larger and larger n such that $R(\sigma) \downarrow$ for all σ with $|\sigma| < n$. Because γ is visited infinitely often, we must have run R on these σ for arbitrarily many steps. Therefore, $\exists \sigma R(\sigma) \uparrow$.

If $f(i) = \infty$, then we must have found infinitely many n such that $R(\sigma) \downarrow$ for all σ with $|\sigma| < n$ and therefore $R(\sigma) \downarrow$. Therefore, we must only verify that in fact $\Psi^A = R$. Let s be the last stage at which γ was injured. If $\Psi^A \neq R$, then at some point we would have seen $\Psi^{\alpha_k}(\sigma) \neq R(\sigma)$ for some α_k we were monitoring. But this would force outcome s_1, which is held permanently if it is ever encountered after the last injury to γ. Thus, we must have $\Psi^A = R$.

□

Lemma 2.7. *Every strategy on the true path is injured at most finitely often.*

Proof. Note from the construction that a strategy can only be injured by strategies of higher priority than it. We go by recursion on f:

The strategy $f \upharpoonright 0$ is the highest priority strategy in the entire construction and cannot be injured by any other strategy.

Suppose $\gamma = f \upharpoonright i$ is a strategy on the true path and $f \upharpoonright j$ is injured at most finitely often for each $j < i$. Any strategy to the left of γ must be visited only finitely often by the definition of the true path and therefore cannot injure γ more than finitely many times. Only finitely many strategies to the left of γ are ever visited, so the total injury caused by strategies to the left of γ is finite. Strategies below or to the right of γ cannot injure γ at all. From the construction, it is clear that each strategy δ causes lower-priority strategies to be injured at most twice for each time δ is injured. For P_e strategies, it occurs when the strategy either picks a new n or sees $\Theta_e(n) \downarrow = 0$. For $L_{\Psi,R}$ strategies, it occurs only when the strategy diagonalizes to ensure a s_1 outcome. Therefore, since each $f \upharpoonright j$ with $j < i$ is

injured at most finitely many times by induction, it must be that γ is injured at most finitely many times. □

Lemma 2.8. *Every strategy on the true path satisfies its requirement.*

Proof. Suppose $\gamma = f \upharpoonright i$ is a strategy on the true path.

If γ is a P_e strategy, then it is successful as long as it manages to keep its witness out of A until an appropriate time and then manages to put it in. Let s be the last stage at which γ is injured. The first time γ is visited after stage s, it picks a new witness larger than any higher priority strategies have. It then forces all lower priority strategies to pick even larger yet witnesses. Therefore, since γ is never injured again, we can be sure no other strategy will put n into A. Also, since γ is on the true path, it will succeed in putting n into A as argued previously.

Now suppose γ is an $L_{\Psi,R}$ strategy. The ∞ outcome is the only one that requires any verification. Suppose therefore that $\Psi^A = R$ with R total. Let s_0 be the last stage at which γ is injured. We must show that the Φ we built to simulate Ψ satisfies the low for speed requirement by showing that it is both correct on all inputs and that there is a polynomial p with $t(\Phi, \sigma) \leq p(t(\Psi^A, \sigma)$ for all σ.

(Correctness) If $\Phi(\sigma)\downarrow = i$, it must be because we saw $\Psi^{\alpha_k}(\sigma)\downarrow = i$ at some stage $s \geq s_0$. γ then imposed the requirement that no $i < k$ could enter A until $R(\sigma)\downarrow = i$. If γ had seen $R(\sigma)\downarrow \neq i$, it would have diagonalized against it by making $\alpha_k \prec A$. Since no injury occurred after s_0, this would have been respected and we would have had $\Psi^A \neq R$. Therefore, we must have had $R(\sigma)\downarrow = i$ and so $\Phi(\sigma) = R(\sigma) = \Psi^A(\sigma)$.

(Fastness) Φ may take as finite data the state of the construction at stage s_0 and therefore know that γ will not ever be injured again. On input σ, Φ computes by recursion on t:

At stage t, in $O(t^2)$ steps compute the $\alpha_{k_1}, \ldots, \alpha_{k_t}$ being monitored by $L_{\Psi,R}$ strategies at stage t. We can do this by simply running the construction and watching for when P_e strategies add new threads to the monitored set. Since at most t strategies act at stage t and each P_e strategy is allowed only t steps, we can do this. Also note that while $L_{\Psi,R}$ strategies take more than t steps to execute (since they are monitoring 2^s strings), these strategies do not affect which threads are being monitored except when they attempt to force the s_1 outcome. In that case, they follow the procedure for adding numbers to A which specifically include a delay that allows Φ to "see" that this is happening before actually doing it.

After identifying the active threads, run $\Psi^{\alpha_{k_i}}(\sigma)$ for t steps each and if $\Phi^{\alpha_{k_i}}(\sigma)\downarrow = i$, set $\Phi(\sigma) = i$ and halt. Then stage t must run up to t simulations of length t each. Since it took $O(t^s)$ stages to identify the threads to use, stage t is therefore $O(t^2 + t^2) = O(t^2)$. By the way the construction works, if $\Psi^A(\sigma)\downarrow$ in $s > s_0$ steps, we will see it at stage s. Therefore, Φ may need to run up to s stages, which takes time

$$\sum_{i=0}^{s}(\text{time to run stage i}) \leq \sum_{i=0}^{s} Ci^2 \leq \sum_{i=0}^{s} Cs^2 \leq Cs^3 \tag{2.2}$$

So we get that indeed Φ is no worse than polynomial time slower than Ψ^A.

□

This concludes the proof of theorem 2.5.

Chapter 3

Connection with Promptly Simple Degrees

3.1 Background

In this chapter, we will investigate the relationship between promptness and low for speed. We begin with some background on promptly simple sets and known equivalences.

Recall that a simple set is one which meets every infinite r.e. set. A promptly simple set is one which not only has at least one element in common with every infinite r.e. set, but also has the property that these elements enter it "reasonably soon" after entering the infinite r.e. set. This is made precise as follows:

Definition 3.1 (Promptly Simple [13])**.** A co-infinite r.e. set A is called *promptly simple* iff there is total recursive function $f : \mathbb{N} \to \mathbb{N}$ such that whenever W_e is infinite, we have

$$\exists s \exists \sigma \left(\sigma \in ((W_e)_{s+1} - (W_e)_s) \ \ \& \ \ \sigma \in A_{f(s)} \right)$$

A degree **a** is said to be promptly simple if **a** contains a promptly simple set. An r.e. set is said to have promptly simple degree if its degree is promptly simple. Ambos-Spies, Jockush, Shore, and Soare proved the following result:

Theorem 3.1 ([1])**.** *Let A be an r.e. set. The following are equivalent:*

1. *A has promptly simple degree*
2. *There is a total recursive function $f : \mathbb{N} \to \mathbb{N}$ with $f(s) \geq s$ such that whenever W_e is infinite we have*

$$\exists \sigma \exists s \left(\sigma \in ((W_e)_{s+1} - (W_e)_s) \ \ \& \ \ A_s \upharpoonright |\sigma| \neq A_{f(s)} \upharpoonright |\sigma| \right)$$

3. *As in (2), but replace $\exists \sigma$ with $\exists^\infty \sigma$.*

Sets satisfying condition (2) above are said to be *promptly permitting*. Perhaps surprisingly, it turns out that there is a deep connection between promptly simple degrees and minimal pairs (see definition 2.1):

Theorem 3.2 ([1], [14])**.** *An r.e. degree* **a** *is promptly simple iff it is not half of a minimal pair of r.e. degrees.*

3.2 Prompt Sets are not Low For Speed

We turn our attention now to the connection between promptly simple sets and sets which are low for speed. In particular, we will show that no prompt r.e. set can be low for speed.

Theorem 3.3. *Let A be promptly permitting and r.e. Then there is a recursive $R \subseteq 2^{<\omega}$ and a recursive functional Ψ such that $\Psi^A = R$ such that for any total recursive functional Φ with $\Phi = R$ and any polynomial p, there is a $\sigma \in 2^{<\omega}$ such that $t(\Phi, \sigma) > p(t(\Psi^A, \sigma))$.*

Proof. Let f be as in theorem 3.1. Let $\{(\Phi_s, p_s)\}_{s\in\omega}$ be an enumeration of all pairs of partial functionals and polynomials with positive integer coefficients. We will construct R, Ψ in stages.

At stage s, we will ensure that one of the following holds:

- $\exists\sigma \Phi_s(\sigma) \neq R(\sigma)$
- $\exists\sigma\ t(\Phi_s, \sigma) > p_s(t(\Psi^A, \sigma))$

In addition we will ensure R is total by defining $R(\sigma)$ by stage $|\sigma|$ at the latest. We will also get $\Psi^A = R$ by ensuring that anytime we define $R(\sigma)$ we likewise define a Ψ computation and also ensure that said computation remains accessible regardless of if more strings enter A in the future.

The construction at stage s is as follows. If $R(\sigma)$ has not yet been defined and $|\sigma| < s$, set $R(\sigma) = \Psi(\sigma) = 0$. Also activate a copy of the following strategy to ensure we meet our requirements for (Φ_s, p_s):

1. Set τ to be the empty string and let W be an initially empty r.e. set we will use to challenge the promptness of A.
2. Pick some σ such that $R(\sigma)$ has not yet been defined.
3. Define $\Psi^B(\sigma) = 0$ for every $B \subseteq 2^{<\omega}$ with $B \upharpoonright l = A_s \upharpoonright l$. Let r be an upper bound on the time required by any of these computations. Note that because of the B's we are using, it is possible that as short strings are later enumerated into A our Ψ computation will no longer apply. We will deal with this in step 6.
4. If $\Phi_s(\sigma) \downarrow = 0$ in less than $p_s(r)$ steps, put τ into W and go to step 5. If not, go to step 6.

5. If $A \upharpoonright |\tau|$ promptly permits (that is, if $A \upharpoonright |\tau|$ changes by step $f(s)$), then set $R(\sigma) = \Psi^B(\sigma) = 1$ for all B with $A_{f(s)} \subseteq B$ and stop. If not, then append a 1 to τ and go back to step 2.

6. Monitor the construction from now on and if in the future we see $A \upharpoonright |\tau|$ change, then go back to step 2.

Lemma 3.4. *Each strategy reaches its step (2) only finitely many times and is therefore finite.*

Proof. Since A is r.e., $A \upharpoonright |\tau|$ can change no more than $2^{|\tau|}$ times for a fixed τ. Therefore, if we reach step (2) infinitely often, it must be because infinitely often A fails to promptly permit. However, since we are assuming A is of promptly simple degree, this is not possible. Therefore, each strategy must be finite. □

Lemma 3.5. *The strategy for Φ_s, p_s succeeds in ensuring that there is some σ such that $(\Phi_s(\sigma) \neq R(\sigma)$ or $t(\Phi_s, \sigma) \geq p_s(t(\Psi^A, \sigma))$.*

Proof. We know the strategy eventually stops reaching step (2), so it must end in either step (5) or step (6).

If it ends in step 5, then we have ensured that $\Psi^A \neq \Phi_s$ since we have defined Ψ such that $\Psi^B(\sigma) = 1$ for any $B \supseteq A_{f(s)}$ and A clearly has this property since it is r.e.

If it ends in step 6, then it must have been because we found a τ and a σ with $t(\Phi_s, \sigma) > p_s(t(\Psi^B, \sigma))$ for all B with $B \upharpoonright |\tau| = A_s \upharpoonright |\tau|$. Because $A \upharpoonright |\tau|$ must not change, we get that the actual Ψ^A computation matches the last one we enumerated in order to diagonalize against R. □

Therefore, we have successfully diagonalized against all possible Φ, p and conclude that A must not be low for speed.

□

Note that the above proof uses the fact that A is r.e. in an essential manner. We leave it open whether the above result holds for all sets of promptly simple degree.

3.3 Low For Speed Differs From Non-Prompt

In light of theorem 3.2, we might be tempted to think that being low for speed is precisely the same as being non-prompt. However, the following theorem shows that this is not the case.

Theorem 3.6. *There is a non-recursive r.e. set $A \subseteq \mathbb{N}$ which is not promptly simple and has the property that any $B \equiv_T A$ is not low for speed.*

Proof. We will give another $0''$-priority argument; this will be our final such priority argument.

Requirements

Our requirements will be as follows:

- For each e, P_e: $A \neq \Phi_e$.
- For each partial recursive g, N_g : If g is total, then there is some infinite r.e. $\mathcal{U}$ such that whenever σ enters $\mathcal{U}$ at stage s, $A_s \upharpoonright |\sigma| = A_{g(s)} \upharpoonright |\sigma|$.
- For every pair of functionals Φ, Γ, $L_{\Phi,\Gamma}$: If Φ^A is total and is some set B such that $\Gamma^B = A$ (ie, if $B \equiv_T A$), then there is some Ψ and some total R such that $\Psi^B = R$ and each of the following sub-requirements holds:
  - For each functional Θ and each polynomial p, $S_{\Phi,\Gamma,\Theta,p}$: There is some σ such that $\Theta(\sigma)$ does not converge to $\Psi^A(\sigma)$ in less than $p(t(\Psi^A, \sigma))$ steps.

Note that the Ψ referenced by the sub-requirements must be shared between all the sub-requirements of the same L requirement.

Strategies

Strategy for P_e

We will use the same basic strategy as we used for theorems 2.5 and 2.1. Namely,

1. Pick some σ not yet in A and larger than anything mentioned in the construction so far.
2. Keep σ out of A until $\Phi_e(\sigma) \downarrow = 0$.
3. Put σ into A and stop.

As usual, these strategies are injured iff σ enters A and we haven't seen $\Phi_e(\sigma) \downarrow = 0$.

Strategy for N_g

At stage s, if this strategy is not waiting for $g(s')$ to converge for some $s' < s$ and if $s > g(s')$ for all the $g(s')$ this strategy has previously calculated, then put 1^s into $\mathcal{U}$ and do not allow $A \upharpoonright s$ to change until we have seen $g(i)$ converge for all $i \leq s$ and have reached stage $g(s)$.

There are two possible outcomes for these strategies:

- ∞ : $g(s)$ always converges.
- w: We find some stage where we restrain $A \upharpoonright s$, but we wait forever for $g(i)$ to converge for some $i \leq s$.

Such a strategy is injured iff some higher priority puts some σ with $|\sigma| \leq s$ into A before stage $g(s)$.

Strategy for $L_{\Phi,\Gamma}$

We will not actually do anything directly for these requirements, and will instead take care of them via a sub-strategies for each of the sub-requirements. However, each strategy will keep track of its own individual Ψ, R (as described in the description of these requirements) and share these with all its sub-strategies.

Strategy for $S_{\Phi,\Gamma,\Theta,p}$

1. Pick some σ not yet in A and longer than anything seen so far and keep σ out of A.
2. Wait until a stage s when we can find an l and an m such that
   a) $|\sigma| < m < s$ and $|\sigma| < l < s$
   b) $B = \Phi^{A_s}(\tau)\downarrow$ whenever $|\tau| \leq l$ and $\Gamma^{B\restriction l}(\tau)\downarrow = A_s(\tau)$ when $|\tau| \leq m$ and the use of any such Γ computations does not exceed l and no computations take more than s steps.
3. Pick some τ such that $R(\tau)$ is not yet defined. Here, R is shared between all the sub-strategies of the same $L_{\Phi,\Gamma}$ requirement.
4. Set $\Psi^{B\restriction l}(\tau) = 0$. Let t_0 be the number of steps required by this computation. Like R, Ψ is shared between all the sub-strategies for the same $L_{\Phi,\Gamma}$ requirement.
5. If $\Theta(\tau)\downarrow = 0$ in no more than $p(t_0)$ steps, put σ into A. Set $\Psi^{B'}(\tau) = 1$ for all B' with $B' \restriction l \neq B \restriction l$ and set $R(\tau) = 1$.
6. Otherwise, set $R(\tau) = 0$ and prohibit any strings of length less than or equal to s from entering A.

The setup described in step (2) captures that idea that A computes $B \restriction l$ and that $B \restriction l$ computes $A \restriction m$. This can be pictured visually as in figure 3.1.

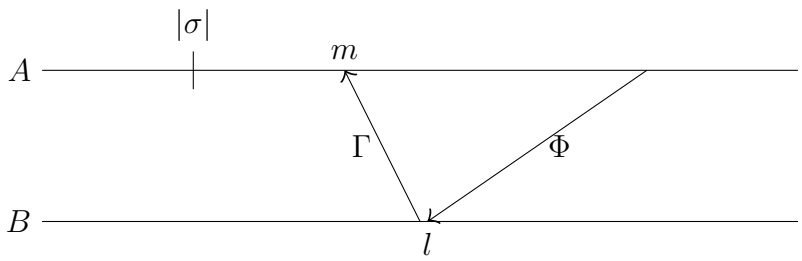

Figure 3.1: A graphical representation of the important setup showing $A \equiv_T B$

There are two possible outcomes for an L strategy based on what happens with its S sub-strategies:

- w: We could wait forever to find a setup as in figure 3.1.
- ∞ : We always find such setups.

Note that once a sub-strategy has found its setup, the remainder of its actions are finite. An S strategy is injured if either the σ it picks in step 1 is put into A without its consent, or if A changes on short strings after reaching step 6.

Tree of strategies and the construction

Order the N, P, and S requirements in some effective way, but do not include the L requirements in this list.

We now build our tree in the usual way (see theorems 2.1 or 2.5). At a level corresponding to an N_g requirement, we add branches labeled ∞ and w, and at levels corresponding to P or S requirements, we add a single successor labeled o. We order the outcomes as $\infty < w < o$.

Our notion of priority will be the same as it has been in previous proofs. Namely, a strategy γ has higher priority than δ iff it lies above or to the left of δ.

Before describing the actions of the individual strategies, we will describe how a strategy can put a string into A. Suppose some strategy wants to put some σ into A at stage s. It can safely ignore any restraints imposed by lower-priority strategies, but it must respect those from higher-priority strategies. With the way we pick strings to consider putting into A, we know that $|\sigma|$ will be larger than any restraint imposed by a strategy to the left of γ. Therefore, we need only consider restraints imposed by strategies above γ. Restraints imposed by any S or P strategies must be respected indefinitely, so γ must pick a new, longer σ whenever these strategies impose restraints. Similarly, restraints imposed by N_g strategies that γ believes have outcome w must be respected indefinitely; again, γ must therefore ensure to pick a long enough σ so that these restraints will not interfere. This leaves only the case where a restraint is imposed by an N_g strategy that γ believes has outcome ∞. In this case, γ can wait until that strategy drops its restraint and then put σ into A. We will need to organize the N_g strategies so that this strategy will not impose a new restraint until all other ∞-outcome N_g strategies above it also drop their restraints. γ therefore will eventually see a window in which to put σ into A before new restraints are imposed.

As usual, our construction will be in stages. At stage s, we traverse the tree up to a depth of s. Let γ_0 be the root of our tree. Now suppose we are currently at strategy γ_i. As usual, we take action based on what type of strategy this is:

(γ_i is a P_e-strategy) If we have not yet picked some σ or have been injured since the last time this strategy was visited, choose a σ longer than anything referenced by the construction so far. With priority γ_i, keep σ out of A. Regardless of whether we have a σ from before or just picked one, run s steps of $\Phi_e(\sigma)$. If it converges and gives 0, put σ into A and stop considering this strategy. Set $\gamma_{i+1} = \gamma_i \hat{} o$.

(γ_i is an N_g-strategy) If we are not currently waiting on $g(m)$ to converge for some $m < s$ that we picked at a previous stage, then put 1^s into $\mathcal{U}$ and with priority γ_i do not allow A to change on strings of length less than or equal to s. Set $m = s$.

Run up to s steps of $g(i)$ for $i \leq m$. If they all converge, then set $\gamma_{i+1} = \gamma_i \hat{} \infty$ and release the restraint on $A \upharpoonright m$ at stage $g(m)$. Until such stage, take no action when this strategy is visited (other than maintaining the restraint on $A \upharpoonright m$). Also, for each $\delta \prec \gamma_i$ that is a $N_{g'}$ strategy, if γ_i believes δ has outcome ∞, do not pick a new m until δ has also converged and dropped its restraint. In this way, we ensure that any strategies below γ_i will have a "window" through which to put strings into A before γ_i imposes a new restraint.

If $g(i)$ does not converge in s steps for some $i \leq m$, then set $\gamma_{i+1} = \gamma_i \hat{} w$ and maintain the restraint on A.

(γ_i is an $S_{\Phi,\Gamma,\Theta,p}$ strategy) If this sub-strategy has not yet been satisfied and has not been visited since its last injury, pick some σ longer than anything mentioned in the construction so far. If this sub-strategy finds its setup as described in step (2) of the sub-strategy description above, allow it to act as described previously, mark it as satisfied. Otherwise, continue to hold σ out of A until a stage where we do find such a setup. In either case, set $\gamma_{i+1} = \gamma_i \hat{} o$.

Repeat the above until we have built γ_s.

Verification

We must now verify that the A built via this construction is as needed for our theorem. Let f_s be the path built at stage s, and as usual let $f = \liminf f_s$ be the leftmost path visited infinitely often.

Lemma 3.7. *f is correct about the outcome of all N_g strategies along it.*

Proof. Let $\gamma = f \upharpoonright i$ be an N_g strategy on f. Then $f(i) = \infty$ iff there are infinitely many m such that $g(i)$ converges for all $i \leq m$. That is, iff g is total. □

We will therefore call f the true path since all other strategies have only one outcome.

Lemma 3.8. *Every strategy on the true path is injured only finitely many times.*

Proof. Let γ be a strategy on f. As in previous proofs, we need only be worried about other strategies on f since those the left of γ are visited only finitely many times and those to the right all have lower priority and therefore cannot injure γ. Note that each strategy puts at most one string into A after the last time it is injured. Therefore, by induction on the depth of the strategies on f, each one causes only finitely many injuries to those below it and so each strategy is in fact injured only finitely many times.

□

Lemma 3.9. *Every strategy on the true path satisfies its (sub-)requirement.*

Proof. Let $\gamma = f \upharpoonright i$ be a strategy on the true path and let s_0 be the last stage at which it is injured.

(γ is a P_e strategy) Since γ is not injured again, it will succeed in keeping σ out of A until it sees $\Phi_e(\sigma) \downarrow = 0$. Therefore, if we never see $\Phi_e(\sigma) \downarrow = 0$, we have ensured $\Phi_e(\sigma) \neq A(\sigma)$.

If at some later stage we do see $\Phi(\sigma)\downarrow = 0$, then this strategy will succeed in putting σ into A as described previously.

(γ is an N_g strategy) Suppose g is total and s_0 is the last stage at which γ is injured. Then since γ is visited infinitely often, it must pick infinitely many m such that γ puts 1^m into $\mathcal{U}$ at stage $s > s_0$ and since it is not injured it must in fact succeed in ensuring that $A_s \upharpoonright m = A_{g(s)} \upharpoonright m$. Therefore, g is not a promptness function for A.

(γ is an $S_{\Phi,\Gamma,\Theta,p}$ strategy) Suppose Φ, Γ are functionals describing some $B \equiv_T A$. That is, Φ^A is total and $\Gamma^{\Phi^A} = A$. Then at some stage $s > s_0$, γ will succeed in finding its setup as described by figure 3.1. If this strategy ends in case (5), then since $|\sigma| < m$, we have that $A \upharpoonright m$ changed when σ went into A. Therefore, we must also have the $B \upharpoonright l$ changes as well or else we would not have $\Gamma^B = A$. Therefore, we would have $\Psi^B(\sigma) = R(\sigma) = 1$ but $\Theta(\sigma) = 0$. If we end in case (6), then we have ensured that $\Theta(\sigma)$ is slower than $\Psi^B(\sigma)$ by more than p. Since $A \upharpoonright s$ does not change after this point, we also have that B cannot change below the use of $\Psi^B(\sigma)$ and so the computation we found remains. In either case, we have diagonalized against Θ, p and therefore satisfied the sub-requirement.

□

Since there is a strategy for each requirement and sub-requirement on f, we conclude that all requirements are satisfied and therefore A is as desired.

□

Chapter 4

Generics and P vs NP

4.1 Background on generics and forcing

The notion of generics and forcing was originally introduced by Cohen in [3] in order to prove the independence of the continuum of hypothesis from ZF. However, as Cohen's notion works for the full generality of set theory, it is far more powerful than we need. The simplification of forcing to first order arithmetic is due to Feferman in [5]. The definitions and results in this section are due to Cohen and Feferman.

At a high level, the general idea is that any fact about a generic is determined by some finite initial segment of G; that is, the truth of that fact is *forced* by some finite piece of G. These finite pieces of information are called *conditions*.

For the remainder of this chapter, conditions will consist of pairs (g, n_g) such that $g \subseteq 2^{<\omega}$ is finite, $n_g \in \mathbb{N}$ and for all $\sigma \in g$ $|\sigma| \leq n_g$. Let $\mathbb{P}$ be the set of all such pairs, and note that $\mathbb{P}$ is recursive. We will write $(h, n_h) \leq_{\mathbb{P}} (g, n_g)$ and say (h, n_h) *extends* (g, n_g) iff $g \subseteq h$ and $\sigma \in h - g \rightarrow |\sigma| > n_g$. Note that $\leq_{\mathbb{P}}$ is a recursive relation. Intuitively, a condition (g, n_g) specifies precisely which strings of length $\leq n_g$ will be in the generic G.

Remark. The above definition of conditions is not the one usually encountered when discussing forcing in arithmetic. Instead, we usually consider conditions to be finite $\sigma \in 2^{<\omega}$ and say $\sigma \leq_{\mathbb{P}} \tau$ iff $\tau \preceq \sigma$ (note the reversal here). A generic then is some $G \subseteq \mathbb{N}$ and conditions specify initial segments of G in the obvious way. Because we will be concerned with algorithmic complexity, we will want $G \subseteq 2^{<\omega}$, not $G \subseteq \mathbb{N}$, in order to avoid excessive use of logarithms in our formulas. Our notion is equivalent to the standard one, and having conditions as above also simplifies some formulas versus using conditions $\rho \in 2^{<2^{<\omega}}$ which would be the other natural way to do it.

The reader familiar with forcing in arithmetic can safely skip ahead to section 4.2 at this point.

We are now ready to discuss the forcing relation. We want to capture the idea that any G which has (g, n_g) as an initial segment must have some property. We first define this syntactically and then show that our definition does in fact accomplish our goal.

Definition 4.1. Let $p = (g, n_g)$ be a condition and let φ be a closed formula in the first order language consisting of a set constant X, constant symbols $\ulcorner\sigma\urcorner$ for each $\sigma \in 2^{<\omega}$, a relation $\in$ that can only be used in formulas of the form $x \in X$, and for each recursive relation R a relation symbol φ_R. We define the relation $p \Vdash \varphi$, pronounced p *forces* φ, inductively on φ:

$$\begin{array}{lcl} p \Vdash \varphi_R(\ulcorner\sigma_1\urcorner, \ldots, \ulcorner\sigma_2\urcorner) & \Leftrightarrow & R(\sigma_1, \ldots, \sigma_2 \\ p \Vdash \psi_1 \vee \psi_2 & \Leftrightarrow & p \Vdash \psi_1 \vee p \Vdash \psi_2 \\ p \Vdash \exists\sigma\psi(\sigma) & \Leftrightarrow & \exists\sigma\ (p \Vdash \psi(\ulcorner\sigma\urcorner)) \\ p \Vdash \ulcorner\sigma\urcorner \in X & \Leftrightarrow & \sigma \in g \\ p \Vdash \neg\psi & \Leftrightarrow & \forall q \leq_{\mathbb{P}} p\ \ \neg(q \Vdash \psi) \end{array}$$

From this definition, it is clear that if $p \Vdash \varphi$ and $q \leq_{\mathbb{P}} p$, then $q \Vdash \varphi$. We turn our attention now to sets of strings.

Definition 4.2. If $A \subseteq 2^{<\omega}$, then we say

- A forces φ ($A \Vdash \varphi$) iff there is some i such that $(A \upharpoonright i, i) \Vdash \varphi$.
- A decides φ iff $A \Vdash \varphi$ or $A \Vdash \neg\varphi$
- φ is true of A and write $A \models \varphi$ iff φ holds when we replace X by A and $\ulcorner\sigma\urcorner$ by σ.
- A is n-generic iff A decides every Σ^0_n sentence.

For generics, forcing is the same as truth, as shown by the following theorem.

Theorem 4.1 ([3], [5], [8])**.** *Let G be n-generic and let φ be a Σ^n_0 formula. Then $G \Vdash \varphi \Leftrightarrow G \models \varphi$.*

Proof. Let G be n-generic and φ be Σ^0_n. We will go by induction on the structure of φ. The definitions for $\Vdash$ and $\models$ coincide in all cases except negation. Therefore, we need only verify that one case:

$$\begin{aligned} G \models \neg\psi &\Leftrightarrow \neg(G \models \psi) \\ &\Leftrightarrow \neg(G \Vdash \psi) \\ &\Leftrightarrow G \Vdash \neg\psi \ \ (\text{ since } G \text{ is } n\text{-generic and must force either } \varphi \text{ or } \neg\varphi) \end{aligned}$$

□

Definition 4.3. We say $A \subseteq \mathbb{P}$ is *dense* iff $\forall p \in \mathbb{P} \exists q \in \mathbb{P}\ (q \leq_{\mathbb{P}} p)$.

Definition 4.4. If $A \subseteq \mathbb{P}$ and $G \subseteq 2^{<\omega}$, we say G *meets* A if there is some n such that $(G \upharpoonright n, n) \in A$.

Theorem 4.2. *Let $G \subseteq 2^{<\omega}$ be n-generic. Then G meets every dense Σ^0_n set D.*

Proof. Let φ_D be a Σ^0_n formula for D. Then the formula $\varphi = \exists i\ \varphi_D(X \upharpoonright i, i)$ is Σ^0_n, and so G decides it. If $G \Vdash \varphi$, then by theorem 4.1, we have the desired result. $G \Vdash \neg\varphi$ is not possible by the fact that D is dense, since given some $p \Vdash \neg\varphi$, there is some $q \leq_{\mathbb{P}} p$ such that $q \in D$ and thus $q \Vdash \varphi$. □

In fact, the above theorem reverses and provides an alternative definition of G being n-generic ([15]).

Remark. For the reader familiar only with forcing from set theory, one can note that these definitions are in fact the same as the more general ones used in that field. Namely, given any $G \subseteq 2^{<\omega}$, it is easy to see that the set $\{(G \upharpoonright i, i) : i \in N\}$ forms a filter and, conversely, that any infinite filter on $\mathbb{P}$ must be of this form.

For more background on generics and forcing, see Chapter XII of [15] or Feferman's original paper [5].

4.2 Connection with P vs NP

Theorem 4.3. *If $\mathcal{P} = \mathcal{NP}$, then every 2-generic is low for speed.*

Proof. Suppose $\mathcal{P} = \mathcal{NP}$ and let G be generic. Suppose also that Ψ, R are such that $\Psi^G = R$ and R is total. Since G is 2-generic, and the statement $\Psi^G = R$ is $\Pi_2(G)$, there is some condition (g, n_g) such that $(g, n_g) \Vdash \Psi^G = R$. We must exhibit a Φ such that $\Phi = \Psi^G$ and $t(\Phi, \sigma) \leq q(t(\Psi^G, \sigma))$. Let A be defined as follows:

$$A = \{\langle \sigma, s, i\rangle : s > n_g \ \ \& \ \ \exists (h, n_h) \leq_{\mathbb{P}} (g, n_g) \ \ \big(n_h < s \ \& \ \Psi^h(\sigma) \downarrow = i \text{ in} \leq s \text{ steps}\big)\} \quad (4.1)$$

Lemma 4.4. *$A \in \mathcal{NP}$ when we consider $|\langle \sigma, s, i\rangle|$ to be $|\sigma| + s + 1$.*

Proof. We will build a verifier machine V and a polynomial p such that:

1. Whenever $\langle \sigma, s, i\rangle \in A$, there is some τ with $|\tau| < p(|\langle \sigma, s, i\rangle|)$ and $V(\langle \sigma, s, i\rangle, \tau) \downarrow = 1$ in time $\leq p(|\langle \sigma, s, i\rangle|)$
2. Whenever $\langle \sigma, s, i\rangle \notin A$, there is no τ such that $V(\langle \sigma, s, i\rangle, \tau) \downarrow = 1$

Our verifier $V(\langle \sigma, s, i\rangle, \tau)$ will interpret τ as a witness (h, n_h) satisfying the definition of A from equation 4.1. It must then check that the encoded condition does in fact work:

After we have verified that $n_g \leq n_h < s$, checking that $|h| < n_h$ and that all $\alpha \in h$ have $|\alpha| < n_h$ takes $O(s^2)$ steps since in the worst case we must check s strings each of length s. To check that $g \subseteq h$ also takes time polynomial in s since for each of the constantly many $\alpha \in g$, we need only verify whether α is one of the $\leq s$ strings in h, each of which has length at most s.

Checking whether $\Psi^h(\sigma)\downarrow = i$ in s steps also takes poly(s) many steps since anytime Ψ queries whether $\alpha \in h$, we can answer in no more than $O(s^2)$ time by iterating through all $\leq s$ strings in h, each of which has length at most s.

V accepts and outputs 1 iff τ satisfies all the requirements of a witness for the definition of A.

Note that V will not accept any τ that does not encode a correct h, and that V will, in polynomial time, accept any that do. V therefore satisfies the requirements of an $\mathcal{NP}$ verification machine and so $A \in \mathcal{NP}$. □

Since $A \in \mathcal{NP}$ and we are assuming $\mathcal{P} = \mathcal{NP}$, we can find a functional Θ and a polynomial p witnessing the fact that $A \in \mathcal{P}$. By our convention that $t(\Psi^h, \sigma) \geq |\sigma|$, we note that $\langle \sigma, s, i\rangle \notin A$ unless $|\sigma| < s$ and therefore we can take p to be a function of s alone. Namely, for all σ, s, i, we may assume $t(\Phi, \langle \sigma, s, i\rangle) \leq p(s)$.

We now define our simulator Φ such that $\Phi = \Psi^G$. We take as finite data the value of $\Psi^G(\sigma)$ for all σ with $|\sigma| \leq n_g$.

On input σ with $|\sigma| > n_g$, define $\Phi(\sigma)$ by induction on s. At stage s, use Θ to check whether $\langle \sigma, s, i\rangle \in A$ for $i = 0, 1$. This takes time $\leq 2p(s)$. If so, output $\Phi(\sigma) = i$ for the correct i. Otherwise, set $s = s + 1$ and try again.

We must verify that this algorithm terminates sufficiently quickly and gives the correct answer. If $\Psi^G(\sigma)\downarrow = i$, in s_0 steps and $|\sigma| > n_g$ (so also $s_0 > n_g$), then $\langle \sigma, s_0, i\rangle \in A$ since the h consisting of g together with the elements of G that are actually queried in this computation satisfies the requirements for membership in A. Therefore, $\Phi(\sigma)$ terminates no later than stage s_0. There is some constant C such that stage i takes $\leq Cp(i)$ steps, so we have:

$$t(\Phi, \sigma) \leq \sum_{i=0}^{s_0} Cp(i) \leq \sum_{i=0}^{s_0} Cp(s_0) \leq C \cdot (s_0 + 1)p(s_0) \tag{4.2}$$

which is polynomial in $s_0 = t(\Psi^G, \sigma)$. It only remains to verify that Φ gives the correct value.

Suppose we set $\Phi(\sigma) = i$ because we saw $\langle \sigma, s, i\rangle \in A$. Then there is some $(h, n_h) \leq (g, n_g)$ such that $\Psi^h(\sigma)\downarrow = i$. Since $(g, n_g) \Vdash \Psi^G = R$, we also have $(h, n_h) \Vdash \Psi^G = R$. Therefore, we must also have $\Psi^h(\sigma) = R(\sigma) = \Psi^G(\sigma)$.

Φ therefore satisfies the low for speed requirements in regards to an arbitrary Ψ, R and so we conclude G must be low for speed. □

If a set G meets every dense recursive set, we will say it is *recursively generic*.

Theorem 4.5. *If $\mathcal{P} \neq \mathcal{NP}$, then every recursively generic set is not low for speed.*

Proof. Suppose $\mathcal{P} \neq \mathcal{NP}$ and G is generic. We must build a Ψ and a recursive R such that $\Psi^G = R$ but for all functionals Φ and all polynomials p, either $\Phi \neq R$ or there is some σ such that $t(\Phi, \sigma) \geq p(t(\Psi^G, \sigma))$.

Let R be some recursive set in $\mathcal{NP} - \mathcal{P}$. For concreteness, one could use SAT ([4], [12]) or any other $\mathcal{NP}$-complete problem. Let V be a verifier machine for R with polynomial time bound p_V. We therefore have the following:

- If $\sigma \in R$, then $\exists \tau$ with $|\tau| < p_V(|\sigma|)$ and $V(\sigma, \tau) \downarrow = 1$ in no more than $p_V(|\sigma)$ steps.
- If $\sigma \notin R$, then $\forall \tau$, $V(\sigma, \tau) \neq 1$.

Our goal will be to encode arbitrarily long $\sigma \in R$ into G in a way so that machines with access to G can determine that $\sigma \in R$ quickly. We will do this by encoding a V-witness τ into G as well. Note that we cannot simply put $\sigma \hat{} \tau$ into G since there are exponentially many extensions of σ of the correct length and so G cannot simply search for one without its running time becoming too long. However, if we ensure that no other strings of length between $|\sigma|$ and $|\sigma \hat{} \tau|$ are in G and that $\sigma \hat{} (\tau \restriction i)$ is in G for all i, then we can find τ by searching bit-by-bit for extensions of σ in G.

Even though G is a fixed given set, the fact that it is generic allows us to "encode" things in it by showing that our encoding forms a dense set. For every functional Φ and every polynomial q, let D^q_Φ be a set of conditions such that $(g, n_g) \in D^q_\Phi$ iff there are σ, τ such that

1. $\sigma \in g$, $|\tau| \leq p_V(|\sigma|)$
2. $\forall \alpha \in 2^{<\omega}$ $(|\sigma| < |\alpha| \leq |\sigma \hat{} \tau| \rightarrow (\alpha \in g \leftrightarrow \exists i \leq |\tau| \, (\alpha = \sigma \hat{} (\tau \restriction i))))$
3. $V(\sigma, \tau) \downarrow = 1$ in no more than $p_V(|\sigma|)$ steps
4. $t(\Phi, \sigma) > q(|\sigma|)$ or $\Phi(\sigma) \neq 1$

Lemma 4.6. *D^q_Φ is a recursive, dense subset of $\mathbb{P}$.*

Proof. **D^q_Φ is recursive:** Since $|\sigma \hat{} \tau| \leq n_g$, all string quantifiers are bounded. Each clause is also clearly recursive since we only need to run boundedly many steps of any computation. Therefore, D^q_Φ is recursive.

D^q_Φ is dense: Suppose $(g, n_g) \in \mathbb{P}$ is given. We must find $(h, n_h) \leq_{\mathbb{P}} (g, n_g)$ in D^q_Φ. Since $R \notin \mathcal{P}$, there is some $\sigma \in R$ with $|\sigma| > n_g$ and $\Phi_s(\sigma)$ does not output 1 after $q(|\sigma|)$ steps.

Let τ be a V-witness to $\sigma \in R$, so $|\tau| < p_V(|\sigma|)$ and $V(\sigma, \tau) = 1$ in less than $p_V(|\sigma|)$ steps. Let $n_h = |\sigma| + |\tau|$ and let $g = h \cup \{\sigma \hat{} (\tau \restriction i) : 0 \leq i \leq |\tau|\}$. Then (h, n_h) is as required.

□

We are now ready to describe our algorithm Ψ^G.

1. On input σ, check if $\sigma \in G$. If so, go to step 2. If not, go to step 3.
2. Attempt to build a V-witness by recursion on i. Let τ_i be our guess at stage i. Start at $i = 0$, set τ_0 to be the empty string and do the following:

a) Check if $V(\sigma, \tau_i) \downarrow = 1$ in no more than $p_V(|\sigma|)$ steps. If so, accept σ and terminate.

b) Check if $\tau_i \hat{} 0 \in G$ or $\tau_i \hat{} 1 \in G$. If neither or both are in G, give up and go to step 3. If exactly one is in G, set $\tau_{i+1} = \tau_i \hat{} j$ for the correct j and increment i.

c) If $i > p_V(|\sigma|)$, give up and go to step 3. Otherwise, go back to step 2a.

3. Run $R(\sigma)$ and output the result.

Since R is recursive and V is a verifier machine for R, we are guaranteed that Ψ^G is total and that $\Psi^G = R$. We now analyze the complexity of our algorithm. Step 1 takes $O(|\sigma|)$ steps. Step 2a takes $O(p_V(|\sigma|))$ steps, 2b takes $O(|\sigma| + i)$ steps, and 2c takes $O(1)$ steps. Since $i \leq p_V(|\sigma|)$, the entirety of step 2 takes time polynomial in $|\sigma|$. Step 3 cold take arbitrarily long. Let q_0 be a polynomial such $t(\Psi^G, \sigma) \leq q_0(|\sigma|)$ for any σ whenever Ψ ends in step 2a.

We are now ready to show that G is not low for speed. Let Φ, p be any functional and polynomial. Let $q = p \circ q_0$. Then D^q_Φ is dense, so there is some $(g, n_g) \in D^q_\Phi$ such that $G \upharpoonright n_g = g$. Let σ, τ be the witness to the fact that $(g, n_g) \in D^q_\Phi$. Then by construction, $\Psi^G(\sigma)$ will terminate in step 2a once it has found τ. By the definition of D^q_Φ, either $\Phi(\sigma)$ is wrong, or we have

$$t(\Phi, \sigma) > q(|\sigma|)) = p(q_0(|\sigma|)) \geq p(t(\Psi^G, \sigma)) \tag{4.3}$$

As Φ, p was arbitrary, we conclude G is not low for speed.

□

Note that the right side of theorem 4.5 is a strengthening of the negation of the right side of theorem 4.3, so we have:

Corollary 4.7. *$\mathcal{P} = \mathcal{NP}$ iff every 2-generic is low for speed*

As shown in [2], there are recursive sets A, B such that $\mathcal{P}^A = \mathcal{NP}^A$ and $\mathcal{P}^B \neq \mathcal{NP}^B$. Note that the proofs of theorems 4.3 and 4.5 both relativize, thus giving the following:

Corollary 4.8. *There are recursive sets A, B such that for every 2-generic G, $G \in \mathbf{LFS}^A$ and $G \notin \mathbf{LFS}^B$.*

Proof. If C is recursive, then being 2-generic is equivalent to being 2-generic relative to C. □

Chapter 5

Conclusions

We have now seen that low for speed provides a robust and interesting notion of lowness. A number of directions for further research present themselves.

First, we could ask what other properties can be combined with low for speed. For example, is it possible to build a measure one set of reals all of which are low for speed? That is, can we combine randomness with low for speed? Alternatively, one could ask if answers to these questions are equivalent to solving classical complexity theoretic questions such as $P = RP$ or $P = BPP$. This is especially interesting in light of Corollary 4.7 where we saw that asking whether generics can be low for speed is equivalent to determining whether $P = NP$.

Second, one could ask various degree-theoretic questions about where low for speed sets can and can not be found. We already saw that every degree contains a set which is not low for speed, and that there is a degree which contains only non-low-for-speed sets. While the notion of being low for speed is clearly not downward closed, one could ask if being non-low-for-speed is in some sense upwards closed; that is, if A has the property that every $B \equiv_T A$ is not low for speed and $C \geq_T A$, is it necessarily true that C is also not low for speed?

Third, nearly any traditional degree-structure question could be rephrased to ask about sets which are low for speed. As an example, are there sets A and B, both low for speed, such that $A \oplus B$ is not low for speed?

Finally, instead of considering speed of computation, one could also consider defining lowness notions for other complexity measures. For example, the study of space bounds has a long and rich history, and one could easily define a notion of low for space and investigate its properties.

References

[1] Klaus Ambos-Spies, Carl G. Jockusch, Richard A. Shore, and Robert I. Soare. An algebraic decomposition of the recursively enumerable degrees and the coincidence of several degree classes with the promptly simple degrees. *Transactions of the American Mathematical Society*, 281(1):pp. 109–128, 1984.

[2] Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

[3] Paul J. Cohen. The independence of the continuum hypothesis. *Proceedings of the National Academy of Sciences of the United States of America*, 50(6):pp. 1143–1148, 1963.

[4] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[5] S. Feferman. Some applications of the notions of forcing and generic sets. *Fund. Math.*, 56:325–345, 1964/1965.

[6] Kurt Gödel. On the length of proofs. In S. Feferman et al, editor, *Kurt Gödel: Collected Works*, volume 1, pages 396–399. Oxford University Press, 1986.

[7] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:pp. 285–306, 1965.

[8] Peter G. Hinman. Some applications of forcing to hierarchy problems in arithmetic. *Mathematical Logic Quarterly*, 15(20-22):341–352, 1969.

[9] Antonin Kuera and Sebastiaan A. Terwijn. Lowness for the class of random sets. *The Journal of Symbolic Logic*, 64(4):pp. 1396–1402, 1999.

[10] A. H. Lachlan. Lower bounds for pairs of recursively enumerable degrees. *Proc. London Math. Soc*, 16:537–569, 1966.

[11] Steffen Lempp. Priority arguments in computability theory, model theory, and complexity theory. `http://www.math.wisc.edu/~lempp/papers/prio.pdf`.

[12] Leonid Levin. Universal search problems. *Problems of Information Transmission*, 9:115–116, 1973.

[13] Wolfgang Maass. Recursively enumerable generic sets. *The Journal of Symbolic Logic*, 47(4):pp. 809–823, 1982.

[14] Wolfgang Maass, Richard Shore, and Michael Stob. Splitting properties and jump classes. *Israel Journal of Mathematics*, 39:210–224, 1981. 10.1007/BF02760850.

[15] Piergiorgio Odifreddi. *Classical Recursion Theory*, volume 2 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science B.V., 1999.

[16] R.J. Parikh. Some results on the length of proofs. *Transactions of the American Mathematical Society*, 177:29–36, 1973.

[17] Emil Post. Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, 50:284–316, 1944.

[18] Robert I. Soare. *Recursively Enumerable Sets and Degrees*. Perspectives in Mathematical Logic. Springer, 1987.

[19] Alan Turing. In response to: Information, machines, and brains by A.M. Uttley. In *Conference on Information theory*, September 1950.

[20] C. E. M. Yates. A minimal pair of recursively enumerable degrees. *The Journal of Symbolic Logic*, 31(2):pp. 159–168, 1966.

[21] Domenico Zambella. On sequences with simple initial segments. *ILLC technical report ML-1990-05*, 1990.