2
Computability Theory

In this chapter we will develop a significant amount of computability theory. Much of this technical material will not be needed until much later in the book, and perhaps in only a small section of the book. We have chosen to gather it in one place for ease of reference. However, as a result this chapter is quite uneven in difficulty, and we strongly recommend that one use most of it as a reference for later chapters, rather than reading through all of it in detail before proceeding. This is especially so for those unfamiliar with more advanced techniques such as priority arguments.

In a few instances we will mention, or sometimes use, methods or results beyond what we introduce in the present chapter. For instance, when we mention the work by Reimann and Slaman [327] on "never continuously random" sets in Section 6.12, we will talk about Borel Determinacy, and in a few places in Chapter 13, some knowledge of forcing in the context of computability theory could be helpful, although our presentations will be self-contained. Such instances will be few and isolated, however, and choices had to be made to keep the length of the book somewhat within reason.

2.1 Computable functions, coding, and the halting problem

At the heart of our understanding of algorithmic randomness is the notion of an *algorithm*. Thus the tools we use are based on classical computability theory. While we expect the reader to have had at least one course in the rudiments of computability theory, such as a typical course on "theory of computation", the goal of this chapter is to give a reasonably self-contained account of the basics, as well as some of the tools we will need. We do, however, assume familiarity with the technical definition of computability via Turing machines (or some equivalent formalism). For more details see, for example, Rogers [334], Salomaa [347], Soare [366], Odifreddi [310, 311], or Cooper [79].

Our initial concern is with functions from A into $\mathbb{N}$ where $A \subseteq \mathbb{N}$, i.e., *partial* functions on $\mathbb{N}$. If $A = \mathbb{N}$ then the function is called *total*. Looking only at $\mathbb{N}$ may seem rather restrictive. For example, later we will be concerned with functions that take the set of finite binary strings or subsets of the rationals as their domains and/or ranges. However, from the point of view of classical computability theory (that is, where resources such as time and memory do not matter), our definitions naturally extend to such functions by *coding*; that is, the domains and ranges of such functions can be coded as subsets of $\mathbb{N}$. For example, the rationals $\mathbb{Q}$ can be coded in $\mathbb{N}$ as follows.

Definition 2.1.1. Let $r \in \mathbb{Q} \setminus \{0\}$ and write $r = (-1)^{\delta}\frac{p}{q}$ with $p, q \in \mathbb{N}$ in lowest terms and $\delta = 0$ or 1. Then define the *Gödel number* of r, denoted by $\#(r)$, as $2^{\delta}3^{p}5^{q}$. Let the Gödel number of 0 be 0.

The function $\#$ is an injection from $\mathbb{Q}$ into $\mathbb{N}$, and given $n \in \mathbb{N}$ we can decide exactly which $r \in \mathbb{Q}$, if any, has $\#(r) = n$. Similarly, if σ is a finite binary string, say $\sigma = a_1a_2\dots a_n$, then we can define $\#(\sigma) = 2^{a_1+1}3^{a_2+1}\dots(p_n)^{a_n+1}$, where p_n denotes the nth prime. There are myriad other codings possible, of course. For instance, one could code the string σ as the binary number 1σ, so that, for example, the string 01001 would correspond to 101001 in binary. Coding methods such as these are called "effective codings", since they include algorithms for deciding the resulting injections, in the sense discussed above for the Gödel numbering of the rationals.

Henceforth, unless otherwise indicated, when we discuss computability issues relating to a class of objects, we will always regard these objects as (implicitly) effectively coded in some way.

Part of the philosophy underlying computability theory is the celebrated Church-Turing Thesis, which states that *the algorithmic (i.e., intuitively computable) partial functions are exactly those that can be computed by Turing machines on the natural numbers.* Thus, we formally adopt the definition of algorithmic, or *computable*, functions as being those that are computable by Turing machines, but argue informally, appealing to the intuitive notion of computability as is usual. Excellent discussions of the subtleties of the Church-Turing Thesis can be found in Odifreddi [310] and Soare [367].

There are certain important basic properties of the algorithmic partial functions that we will use throughout the book, often implicitly. Following the usual computability theoretic terminology, we refer to such functions as *partial computable functions*.

Proposition 2.1.2 (Enumeration Theorem – Universal Turing Machine)**.** *There is an algorithmic way of enumerating all the partial computable functions. That is, there is a list $\Phi_0, \Phi_1, \ldots$ of all such functions such that we have an algorithmic procedure for passing from an index i to a Turing machine computing Φ_i, and vice versa. Using such a list, we can define a partial computable function $f(x, y)$ of two variables such that $f(x, y) = \Phi_x(y)$ for all x, y. Such a function, and any Turing machine that computes it, are called* universal.

To a modern computer scientist, this result is obvious. That is, given a program in some computer language, we can convert it into ASCII code, and treat it as a number. Given such a binary number, we can decode it and decide whether it corresponds to the code of a program, and if so execute this program. Thus a compiler for the given language can be used to produce a universal program.

Henceforth, we fix an effective listing $\Phi_0, \Phi_1, \ldots$ of the partial computable functions as above. We have an algorithmic procedure for passing from an index i to a Turing machine M_i computing Φ_i, and we identify each Φ_i with M_i.

For any partial computable function f, there are infinitely many ways to compute f. If Φ_y is one such algorithm for computing f, we say that y is an *index* for f.

The point of Proposition 2.1.2 is that we can pretend that we have all the machines $\Phi_1, \Phi_2, \ldots$ in front of us. For instance, to compute 10 steps in the computation of the 3rd machine on input 20, we can pretend to walk to the 3rd machine, put 20 on the tape, and run it for 10 steps (we write the result as $\Phi_3(20)[10]$). Thus we can *computably* simulate the action of computable functions. In many ways, Proposition 2.1.2 is the platform that makes undecidability proofs work, since it allows us to diagonalize over the class of partial computable functions *without leaving this class*. For instance, we have the following result, where we write $\Phi_x(y)\downarrow$ to mean that Φ_x is defined on y, or equivalently, that the corresponding Turing machine halts on input y, and $\Phi_x(y)\uparrow$ to mean that Φ_x is not defined on y.

Proposition 2.1.3 (Unsolvability of the halting problem)**.** *There is no algorithm that, given x, y, decides whether $\Phi_x(y)\downarrow$. Indeed, there is no algorithm to decide whether $\Phi_x(x)\downarrow$.*

Proof. Suppose such an algorithm exists. Then by Proposition 2.1.2, it follows that the following function g is (total) computable:

$$g(x) = \begin{cases} 1 & \text{if } \Phi_x(x)\uparrow \\ \Phi_x(x)+1 & \text{if } \Phi_x(x)\downarrow . \end{cases}$$

Again using Proposition 2.1.2, there is a y with $g = \Phi_y$. Since g is total, $g(y)\downarrow$, so $\Phi_y(y)\downarrow$, and hence $g(y) = \Phi_y(y) + 1 = g(y) + 1$, which is a contradiction. □

Note that we *can* define a *partial* computable function g via $g(x) = \Phi_x(x) + 1$ and avoid contradiction, as it will follow that, for any index y for g, we have $\Phi_y(y)\uparrow = g(y)\uparrow$. Also, the reason for the use of *partial* computable functions in Proposition 2.1.2 is now clear: The argument above shows that there is no computable procedure to enumerate all (and only) the total computable functions.

Proposition 2.1.3 can be used to show that many problems are algorithmically unsolvable by "coding" the halting problem into these problems. For example, we have the following result.

Proposition 2.1.4. *There is no algorithm to decide whether the domain of Φ_x is empty.*

To prove this proposition, we need a lemma, known as the s-m-n theorem. We state it for unary functions, but it holds for n-ary ones as well. Strictly speaking, the lemma below is the s-1-1 theorem. For the full statement and proof of the s-m-n theorem, see [366].

Lemma 2.1.5 (The s-m-n Theorem). *Let $g(x, y)$ be a partial computable function of two variables. Then there is a computable function s of one variable such that, for all x, y,*

$$\Phi_{s(x)}(y) = g(x, y).$$

Proof. Given a Turing machine M computing g and a number x, we can build a Turing machine N that on input y simulates the action of writing the pair (x, y) on M's input tape and running M. We can then find an index $s(x)$ for the function computed by N. □

Proof of Proposition 2.1.4. We code the halting problem into the problem of deciding whether $\text{dom}(\Phi_x) = \emptyset$. That is, we show that *if* we could decide whether $\text{dom}(\Phi_x) = \emptyset$ *then* we could solve the halting problem. Define a partial computable function of two variables by

$$g(x, y) = \begin{cases} 1 & \text{if } \Phi_x(x)\downarrow \\ \uparrow & \text{if } \Phi_x(x)\uparrow . \end{cases}$$

Notice that g ignores its second input.

Via the *s*-*m*-*n* theorem, we can consider $g(x, y)$ as a computable collection of partial computable functions. That is, there is a computable s such that, for all x, y,

$$\Phi_{s(x)}(y) = g(x, y).$$

Now

$$\mathrm{dom}(\Phi_{s(x)}) = \begin{cases} \mathbb{N} & \text{if } \Phi_x(x)\downarrow \\ \emptyset & \text{if } \Phi_x(x)\uparrow, \end{cases}$$

so if we could decide for a given x whether $\Phi_{s(x)}$ has empty domain, then we could solve the halting problem. □

We denote the result of running the Turing machine corresponding to Φ_e for s many steps on input x by $\Phi_e(x)[s]$. This value can of course be either defined, in which case we write $\Phi_e(x)[s]\downarrow$, or undefined, in which case we write $\Phi_e(x)[s]\uparrow$.

We will often be interested in the computability of families of functions. We say that $f_0, f_1, \ldots$ are *uniformly (partial) computable* if there is a (partial) computable function f of two variables such that $f(n, x) = f_n(x)$ for all n and x. It is not hard to see that $f_0, f_1, \ldots$ are uniformly partial computable iff there is a computable g such that $f_n = \Phi_{g(n)}$ for all n.

2.2 Computable enumerability and Rice's Theorem

We now show that the reasoning used in the proof of Proposition 2.1.4 can be pushed much further. First we wish to regard all problems as coded by subsets of $\mathbb{N}$. For example, the halting problem can be coded by

$$\emptyset' = \{x : \Phi_x(x)\downarrow\}$$

(or if we insist on the two-variable formulation, by $\{\langle x, y\rangle : \Phi_x(y)\downarrow\}$). Next we need some terminology.

Definition 2.2.1. A set $A \subseteq \mathbb{N}$ is called

(i) *computably enumerable* (*c.e.*) if $A = \mathrm{dom}(\Phi_e)$ for some e, and

(ii) *computable* if A and $\overline{A} = \mathbb{N} \setminus A$ are both computably enumerable.

A set is *co-c.e.* if its complement is c.e. Thus a set is computable iff it is both c.e. and co-c.e. Of course, it also makes sense to say that A is computable if its characteristic function χ_A is computable, particularly since, as mentioned in Chapter 1, we identify sets with their characteristic functions. It is straightforward to check that A is computable in the sense of Definition 2.2.1 if and only if χ_A is computable.

We let W_e denote the eth computably enumerable set, that is, $\mathrm{dom}(\Phi_e)$, and let $W_e[s] = \{x \leqslant s : \Phi_e(x)[s]\downarrow\}$. We sometimes write $W_e[s]$ as $W_{e,s}$.

We think of $W_e[s]$ as the result of performing s steps in the enumeration of W_e.

An *index* for a c.e. set A is an e such that $W_e = A$.

We say that a family of sets $A_0, A_1, \ldots$ is *uniformly computably enumerable* if $A_n = \mathrm{dom}(f_n)$ for a family $f_0, f_1, \ldots$ of uniformly partial computable functions. It is easy to see that this condition is equivalent to saying that there is a computable g such that $A_n = W_{g(n)}$ for all n, or that there is a c.e. set A such that $A_n = \{x : \langle n, x\rangle \in A\}$ for all n. A family of sets $A_0, A_1, \ldots$ is *uniformly computable* if the functions $\chi_{A_0}, \chi_{A_1}, \ldots$ are uniformly computable, which is equivalent to saying that both $A_0, A_1, \ldots$ and $\overline{A_0}, \overline{A_1}, \ldots$ are uniformly c.e.

Definition 2.2.1 suggests that one way to make a set A noncomputable is by ensuring that A is coinfinite and for all e, if W_e is infinite then $A \cap W_e \neq \emptyset$. A c.e. set A with these properties is called *simple*. An infinite set that contains no infinite c.e. set is called *immune*. (So a simple set is a c.e. set whose complement is immune.) Not all noncomputable c.e. sets are simple, since given any noncomputable c.e. set A, the set $\{2n : n \in A\}$ is also c.e. and noncomputable, but is not simple.

The name *computably enumerable* comes from a notion of "effectively countable", via the following characterization, whose proof is straightforward.

Proposition 2.2.2. *A set A is computably enumerable iff either $A = \emptyset$ or there is a total computable function f from $\mathbb{N}$ onto A. (If A is infinite then f can be chosen to be injective.)*

Thus we can think of an infinite computably enumerable set as an effectively infinite list (but *not necessarily in increasing numerical order*). Note that computable sets correspond to decidable questions, since if A is computable, then either $A \in \{\emptyset, \mathbb{N}\}$ or we can decide whether $x \in A$ as follows. Let f and g be computable functions such that $f(\mathbb{N}) = A$ and $g(\mathbb{N}) = \overline{A}$. Now enumerate $f(0), g(0), f(1), g(1), \ldots$ until x occurs (as it must). If x occurs in the range of f, then $x \in A$; if it occurs in the range of g, then $x \notin A$.

It is straightforward to show that $\emptyset'$ is computably enumerable. Thus, by Proposition 2.1.3, it is an example of a computably enumerable set that is not computable. As we will show in Proposition 2.4.5, $\emptyset'$ is a *complete* computably enumerable set, in the sense that for any c.e. set A, there is an algorithm for computing A using $\emptyset'$. We will introduce another "highly knowledgeable" real, closely related to $\emptyset'$ and denoted by Ω, in Definition 3.13.6. Calude and Chaitin [48] pointed out that, in 1927, Émile Borel prefigured the idea of such knowledgeable reals by "defining" a real B such that the nth bit of B answers the nth question in an enumeration of all yes/no questions one can write down in French.

If A is c.e., then it clearly has a *computable approximation*, that is, a uniformly computable family $\{A_s\}_{s\in\omega}$ of sets such that $A(n) = \lim_s A_s(n)$ for all n. (For example, $\{W_e[s]\}_{s\in\omega}$ is a computable approximation of W_e.) In Section 2.6, we will give an exact characterization of the sets that have computable approximations. In the particular case of c.e. sets, we can choose the A_s so that $A_0 \subseteq A_1 \subseteq A_2 \subseteq \cdots$. In the constructions we discuss, whenever we are given a c.e. set, we assume we have such an approximation, and think of A_s as the set of numbers put into A by stage s of the construction. In general, whenever we have an object X that is being approximated during a construction, we denote the stage s approximation to X by $X[s]$.

An *index set* is a set A such that if $x \in A$ and $\Phi_x = \Phi_y$ then $y \in A$. For example, $\{x : \mathrm{dom}(\Phi_x) = \emptyset\}$ is an index set. An index set can be thought of as coding a problem about computable functions (like the emptiness of domain problem) whose answer does not depend on the particular algorithm used to compute a function. Generalizing Proposition 2.1.4, we have the following result, which shows that nontrivial index sets are never computable. Its proof is very similar to that of Proposition 2.1.4.

Theorem 2.2.3 (Rice's Theorem [332]). *An index set A is computable (and so the problem it codes is decidable) iff $A = \mathbb{N}$ or $A = \emptyset$.*

Proof. Let $A \notin \{\emptyset, \mathbb{N}\}$ be an index set. Let e be such that $\mathrm{dom}(\Phi_e) = \emptyset$. We may assume without loss of generality that $e \in \overline{A}$ (the case $e \in A$ being symmetric). Fix $i \in A$. By the *s-m-n* theorem, there is a computable $s(x)$ such that, for all $y \in \mathbb{N}$,

$$\Phi_{s(x)}(y) = \begin{cases} \Phi_i(y) & \text{if } \Phi_x(x)\downarrow \\ \uparrow & \text{if } \Phi_x(x)\uparrow . \end{cases}$$

If $\Phi_x(x)\downarrow$ then $\Phi_{s(x)} = \Phi_i$ and so $s(x) \in A$, while if $\Phi_x(x)\uparrow$ then $\Phi_{s(x)} = \Phi_e$ and so $s(x) \notin A$. Thus, if A were computable, $\emptyset'$ would also be computable. □

Of course, many nontrivial decision problems (such as the problem of deciding whether a natural number is prime, say) are not coded by index sets, and so can have decidable solutions.

2.3 The Recursion Theorem

Kleene's Recursion Theorem (also known as the Fixed Point Theorem) is a fundamental result in classical computability theory. It allows us to use an index for a computable function or c.e. set that we are building in a construction as part of that very construction. Thus it forms the theoretical underpinning of the common programming practice of having a routine make recursive calls to itself.

Theorem 2.3.1 (Recursion Theorem, Kleene [209]). *Let f be a total computable function. Then there is a number n, called a* fixed point *of f, such* *that*

$$\Phi_n = \Phi_{f(n)},$$

and hence

$$W_n = W_{f(n)}.$$

Furthermore, such an n can be computed from an index for f.

Proof. First define a total computable function d via the *s-m-n* Theorem so that

$$\Phi_{d(e)}(k) = \begin{cases} \Phi_{\Phi_e(e)}(k) & \text{if } \Phi_e(e)\downarrow \\ \uparrow & \text{if } \Phi_e(e)\uparrow . \end{cases}$$

Let i be such that

$$\Phi_i = f \circ d$$

and let $n = d(i)$. Notice that Φ_i is total. The following calculation shows that n is a fixed point of f.

$$\Phi_n = \Phi_{d(i)} = \Phi_{\Phi_i(i)} = \Phi_{f\circ d(i)} = \Phi_{f(n)}.$$

The explicit definition of n given above can clearly be carried out computably given an index for f. □

A longer but more perspicuous proof of the recursion theorem was given by Owings [312]; see also Soare [366, pp. 36–37].

There are many variations on the theme of the recursion theorem, such as the following one, which we will use several times below.

Theorem 2.3.2 (Recursion Theorem with Parameters, Kleene [209])**.** *Let f be a total computable function of two variables. Then there is a total computable function h such that $\Phi_{h(y)} = \Phi_{f(h(y),y)}$, and hence $W_{h(y)} = W_{f(h(y),y)}$, for all y. Furthermore, an index for h can be obtained effectively from an index for f.*

Proof. The proof is similar to that of the recursion theorem. Let d be a total computable function such that

$$\Phi_{d(x,y)}(k) = \begin{cases} \Phi_{\Phi_x(\langle x,y\rangle)}(k) & \text{if } \Phi_x(\langle x,y\rangle)\downarrow \\ \uparrow & \text{if } \Phi_x(\langle x,y\rangle)\uparrow . \end{cases}$$

Let i be such that

$$\Phi_i(\langle x,y\rangle) = f(d(x,y),y)$$

for all x and y, and let $h(y) = d(i,y)$. Then

$$\Phi_{h(y)} = \Phi_{d(i,y)} = \Phi_{\Phi_i(\langle i,y\rangle)} = \Phi_{f(d(i,y),y)} = \Phi_{f(h(y),y)}$$

for all y. The explicit definition of h given above can clearly be carried out computably given an index for f. □

It is straightforward to modify the above proof to show that if f is a partial computable function of two variables, then there is a total computable function h such that $\Phi_{h(y)} = \Phi_{f(h(y),y)}$ for all y such that $f(h(y), y)\downarrow$.

In Section 3.5, we will see that there is a version of the recursion theorem for functions computed by *prefix-free machines*, a class of machines that will play a key role in this book.

Here is a very simple application of the recursion theorem. We show that $\emptyset'$ is not an index set. Let f be a computable function such that $\Phi_{f(n)}(n)\downarrow$ and $\Phi_{f(n)}(m)\uparrow$ for all $m \neq n$. Let n be a fixed point for f, so that $\Phi_n = \Phi_{f(n)}$. Let $m \neq n$ be another index for Φ_n. Then $\Phi_n(n)\downarrow$ and hence $n \in \emptyset'$, but $\Phi_m(m)\uparrow$ and hence $m \notin \emptyset'$. So $\emptyset'$ is not an index set. Note that this example also shows that there is a Turing machine that halts only on its own index.

The following is another useful application of the recursion theorem.

Theorem 2.3.3 (Slowdown Lemma, Ambos-Spies, Jockusch, Shore, and Soare [7]). *Let $\{U_{e,s}\}_{e,s\in\omega}$ be a computable sequence of finite sets such that $U_{e,s} \subseteq U_{e,s+1}$ for all e and s. Let $U_e = \bigcup_s U_{e,s}$. There is a computable function g such that for all e, s, n, we have $W_{g(e)} = U_e$ and if $n \notin U_{e,s}$ then $n \notin W_{g(e),s+1}$.*

Proof. Let f be a computable function such that $W_{f(i,e)}$ behaves as follows. Given n, look for a least s such that $n \in U_{e,s}$. If such an s is found, ask whether $n \in W_{i,s}$. If not, then enumerate n into $W_{f(i,e)}$. By the recursion theorem with parameters, there is a computable function g such that $W_{g(e)} = W_{f(g(e),e)}$ for every e. If $n \notin U_{e,s}$, then $n \notin W_{g(e)}$. If $n \in U_{e,s}$ then for the least such s it cannot be the case that $n \in W_{g(e),s}$, since in that case we would have $n \notin W_{f(g(e),e)} = W_{g(e)}$. So $n \in W_{f(g(e),e)} = W_{g(e)}$ but $n \notin W_{g(e),s}$. ☐

We will provide the details of applications of versions of the recursion theorem for several results in this chapter, but will assume familiarity with their use elsewhere in the book.

2.4 Reductions

The key concept used in the proof of Rice's Theorem is that of *reduction*, that is, the idea that "if we can do B then this ability also allows us to do A". In other words, *questions about problem A are* reducible *to ones about problem B*. We want to use this idea to define partial orderings, known as *reducibilities*, that calibrate problems according to computational difficulty. The idea is to have $A \leqslant B$ if the ability to solve B allows us also to solve A, meaning that B is "at least as hard as" A. In this section, we introduce several ways to formalize this notion, beginning with the best-known one, Turing reducibility. For any reducibility $\leqslant_{\mathrm{R}}$, we write $A \equiv_{\mathrm{R}} B$, and say that A and B are *R-equivalent*, if $A \leqslant_{\mathrm{R}} B$ and $B \leqslant_{\mathrm{R}} A$. We write $A <_{\mathrm{R}} B$ if $A \leqslant_{\mathrm{R}} B$ and $B \nleqslant_{\mathrm{R}} A$. Finally, we write $A \mid_{\mathrm{R}} B$ if $A \nleqslant_{\mathrm{R}} B$ and $B \nleqslant_{\mathrm{R}} A$.

2.4.1 Oracle machines and Turing reducibility

An *oracle (Turing) machine* is a Turing machine with an extra infinite read-only *oracle tape*, which it can access one bit at a time while performing its computation. If there is an oracle machine M that computes the set A when its oracle tape codes the set B, then we say that A is *Turing reducible* to B, or *B-computable*, or *computable in B*, and write $A \leqslant_{\mathrm{T}} B$.[1] Note that, in computing $A(n)$ for any given n, the machine M can make only finitely many queries to the oracle tape; in other words, it can access the value of $B(m)$ for at most finitely many m. The definition of relative computability can be extended to functions in the obvious way. We will also consider situations in which the oracle tape codes a finite string. In that case, if the machine attempts to make any queries beyond the length of the string, the computation automatically diverges. All the notation we introduce below for oracle tapes coding sets applies to strings as well.

For example, let $E = \{x : \mathrm{dom}(\Phi_x) \neq \emptyset\}$. In the proof of Proposition 2.1.4, we showed that $\emptyset' \leqslant_{\mathrm{T}} E$. Indeed the proof of Rice's Theorem demonstrates that, for a nontrivial index set I, we always have $\emptyset' \leqslant_{\mathrm{T}} I$. On the other hand, the unsolvability of the halting problem implies that $\emptyset' \nleqslant_{\mathrm{T}} \emptyset$. (Note that $X \leqslant_{\mathrm{T}} \emptyset$ iff X is computable. Indeed, if Y is computable, then $X \leqslant_{\mathrm{T}} Y$ iff X is computable.)

It is not hard to check that Turing reducibility is transitive and reflexive, and thus is a preordering on the subsets of $\mathbb{N}$. The equivalence classes of the form $\deg(A) = \{B : B \equiv_{\mathrm{T}} A\}$ code a notion of equicomputability and are called *Turing degrees (of unsolvability)*, though we often refer to them simply as *degrees*. We always use boldface lowercase letters such as $\mathbf{a}$ for Turing degrees. A Turing degree is *computably enumerable* if it contains a computably enumerable set (which does not imply that all the sets in the degree are c.e.). The Turing degrees inherit a natural ordering from Turing reducibility: $\mathbf{a} \leqslant \mathbf{b}$ iff $A \leqslant_{\mathrm{T}} B$ for some (or equivalently all) $A \in \mathbf{a}$ and $B \in \mathbf{b}$. We will relentlessly mix notation by writing, for example, $A <_{\mathrm{T}} \mathbf{b}$, for a set A and a degree $\mathbf{b}$, to mean that $A <_{\mathrm{T}} B$ for some (or equivalently all) $B \in \mathbf{b}$.

The Turing degrees form an uppersemilattice. The join operation is induced by $\oplus$, where $A \oplus B = \{2n : n \in A\} \cup \{2n+1 : n \in B\}$. Clearly $A, B \leqslant_{\mathrm{T}} A \oplus B$, and if $A, B \leqslant_{\mathrm{T}} C$, then $A \oplus B \leqslant_{\mathrm{T}} C$. Furthermore, if

[1] We can also put resource bounds on our procedures. For example, if we count steps and ask that computations halt in a polynomial (in the length of the input) number of steps, then we arrive at the polynomial time computable functions and the notion of polynomial time (Turing) reducibility. We will not consider such reducibilities here; see Ambos-Spies and Mayordomo [10].

$A \equiv_T \widehat{A}$ and $B \equiv_T \widehat{B}$, then $A \oplus B \equiv_T \widehat{A} \oplus \widehat{B}$. Thus it makes sense to define the *join* $\mathbf{a} \vee \mathbf{b}$ of the degrees $\mathbf{a}$ and $\mathbf{b}$ to be the degree of $A \oplus B$ for some (or equivalently all) $A \in \mathbf{a}$ and $B \in \mathbf{b}$. Kleene and Post [210] showed that the Turing degrees are not a lattice. That is, not every pair of degrees has a greatest lower bound.

For sets $A_0, A_1, \ldots$, let $\bigoplus_i A_i = \{\langle i, n\rangle : n \in A_i\}$. Note that, while $A_j \leqslant_T \bigoplus_i A_i$ for all j, the degree of $\bigoplus_i A_i$ may be much greater than the degrees of the A_i's. For instance, for any function f, we can let $A_i = \{f(i)\}$, in which case each A_i is a singleton, and hence computable, but $\bigoplus_i A_i \equiv_T f$.

We let $\mathbf{0}$ denote the degree of the computable sets. Note that each degree is countable and has only countably many predecessors (since there are only countably many oracle machines), so there are continuum many degrees.

For an oracle machine Φ, we write Φ^A for the function computed by Φ with oracle A (i.e., with A coded into its oracle tape). The analog of Proposition 2.1.2 holds for oracle machines. That is, there is an effective enumeration $\Phi_0, \Phi_1, \ldots$ of all oracle machines, and a *universal oracle (Turing) machine* Φ such that $\Phi^A(x, y) = \Phi_x^A(y)$ for all x, y and all oracles A.

We had previously defined Φ_e to be the eth partial computable function. However, we have already identified partial computable functions with Turing machines, and we can regard normal Turing machines as oracle machines with empty oracle tape; in fact it is convenient to identify the eth partial computable function Φ_e with $\Phi_e^{\emptyset}$. Thus there is no real conflict between the two notations, and we will not worry about the double meaning of Φ_e.

When a set A has a computable approximation $\{A_s\}_{s\in\omega}$, we write $\Phi_e^A(n)[s]$ to mean the result of running Φ_e with oracle A_s on input n for s many steps.

We also think of oracle machines as determining *(Turing) functionals*, that is, partial functions from 2^ω to 2^ω (or ω^ω). The value of the functional Φ on A is Φ^A.

The *use* of a converging oracle computation $\Phi^A(n)$ is $x+1$ for the largest number x such that the value of $A(x)$ is queried during the computation. (If no such value is queried, then the use of the computation is 0.) We denote this use by $\varphi^A(n)$. In general, when we have an oracle computation represented by an uppercase Greek letter, its use function is represented by the corresponding lowercase Greek letter. Normally, we do not care about the exact position of the largest bit queried during a computation, and can replace the exact use function by any function that is at least as large. Furthermore, we may assume that an oracle machine cannot query its oracle's nth bit before stage n. So we typically adopt the following useful conventions on a use function φ^A.

1. The use function is strictly increasing where defined, that is, $\varphi^A(m) < \varphi^A(n)$ for all $m < n$ such that both these values are defined, and

similarly, when A is being approximated, $\varphi^A(m)[s] < \varphi^A(n)[s]$ for all $m < n$ and s such that both these values are defined.

2. When A is being approximated, $\varphi^A(n)[s] \leqslant \varphi^A(n)[t]$ for all n and $s < t$ such that both these values are defined.

3. $\varphi^A(n)[s] \leqslant s$ for all n and s such that this value is defined.

Although in a sense trivial, the following principle is quite important.

Proposition 2.4.1 (Use Principle). *Let $\Phi^A(n)$ be a converging oracle computation, and let B be a set such that $B \upharpoonright \varphi^A(n) = A \upharpoonright \varphi^A(n)$. Then $\Phi^B(n) = \Phi^A(n)$.*

Proof. The sets A and B give the same answers to all questions asked during the relevant computations, so the results must be the same. □

One important consequence of the use principle is that if A is c.e. and Φ^A is total, then A can compute the function f defined by letting $f(n)$ be the least s by which the computation of $\Phi^A(n)$ has settled, i.e., $\Phi^A(n)[t]\downarrow = \Phi^A(n)[s]\downarrow$ for all $t > s$. The reason is that $f(n)$ is the least s such that $\Phi^A(n)[s]\downarrow$ and $A \upharpoonright \varphi^A(n)[s] = A_s \upharpoonright \varphi^A(n)[s]$.

For a set A, let

$$A' = \{e : \Phi_e^A(e)\downarrow\}.$$

The set A' represents the halting problem *relativized* to A. The general process of extending a definition or result in the non-oracle case to the oracle case is known as *relativization.* For instance, Proposition 2.1.3 (the unsolvability of the halting problem) can be relativized with a completely analogous proof to show that $A' \nleqslant_T A$ for all A. Two good (but not hard) exercises are to show that $A <_T A'$ and that if $A \leqslant_T B$ then $A' \leqslant_T B'$. Another important example of relativization is the concept of a set B being *computably enumerable in* a set A, which means that $B = \operatorname{dom}(\Phi_e^A)$ for some e. Most results in computability theory can be relativized in a completely straightforward way, and we freely use the relativized versions of theorems proved below when needed.

2.4.2 The jump operator and jump classes

We often refer to A' as the *(Turing) jump* of A. The *jump operator* is the function $A \mapsto A'$. The *nth jump* of A, written as $A^{(n)}$, is the result of applying the jump operator n times to A. So, for example, $A^{(2)} = A''$ and $A^{(3)} = A'''$. If $\mathbf{a} = \deg(A)$ then we write $\mathbf{a}'$ for $\deg(A')$, and similarly for the nth jump notation. This definition makes sense because $A \equiv_T B$ implies $A' \equiv_T B'$. Note that we have a hierarchy of degrees $\mathbf{0} < \mathbf{0}' < \mathbf{0}'' < \cdots$.

We also define the *ω-jump* of A as $A^{(\omega)} = \bigoplus_n A^{(n)}$. (We could continue to iterate the jump to define α-jumps for higher ordinals α, but we will not need these in this book.)

The halting problem, and hence the jump operator, play a fundamental role in much of computability theory. Closely connected to the jump operator, and also of great importance in computability theory, are the following *jump classes*.

Definition 2.4.2. A set A is *low$_n$* if $A^{(n)} \equiv_{\mathrm{T}} \emptyset^{(n)}$. The low$_1$ sets are called *low*.

A set $A \leqslant_{\mathrm{T}} \emptyset'$ is *high$_n$* if $A^{(n)} \equiv_{\mathrm{T}} \emptyset^{(n+1)}$. The high$_1$ sets are called *high*. More generally, we call an arbitrary set A high if $A' \geqslant_{\mathrm{T}} \emptyset''$.

These classes are particularly well suited to studying $\emptyset'$-computable sets. The following classes are sometimes better suited to the general case.

Definition 2.4.3. A set A is *generalized low$_n$* (*GL$_n$*) if $A^{(n)} \equiv_{\mathrm{T}} (A \oplus \emptyset')^{(n-1)}$.

A set A is *generalized high$_n$* (*GH$_n$*) if $A^{(n)} \equiv_{\mathrm{T}} (A \oplus \emptyset')^{(n)}$.

Note that if $A \leqslant_{\mathrm{T}} \emptyset'$, then A is GL$_n$ iff it is low$_n$, and GH$_n$ iff it is high$_n$. A degree is low if the sets it contains are low, and similarly for other jump classes.

While jump classes are defined in terms of the jump operator, they often have more “combinatorial” characterizations. For example, we will see in Theorem 2.23.7 that a set A is high iff there is an A-computable function that dominates all computable functions, where f *dominates* g if $f(n) \geqslant g(n)$ for all sufficiently large n.

2.4.3 Strong reducibilities

The reduction used in the proof of Rice's Theorem is of a particularly strong type, since to decide whether $x \in \emptyset'$, we simply compute $s(x)$ and ask whether $s(x) \in A$. Considering this kind of reduction leads to the following definition.

Definition 2.4.4. We say that A is *many-one reducible* (*m-reducible*) to B, and write $A \leqslant_{\mathrm{m}} B$, if there is a total computable function f such that for all x, we have $x \in A$ iff $f(x) \in B$.[2]

If B is $\emptyset$ or $\mathbb{N}$ then the only set m-reducible to B is B itself, but we will ignore these cases of the above definition. If the function f in the definition of m-reduction is injective, then we say that A is 1-*reducible* to B, and

[2]In the context of complexity theory, resource bounded versions of m-reducibility are at the basis of virtually all modern NP-completeness proofs. Although Cook's original definition of NP-completeness was in terms of polynomial time *Turing* reducibility, the Karp version in terms of polynomial time m-reducibility is most often used. It is still an open question of structural complexity theory whether there is a set A such that the polynomial time Turing degree of A collapses to a single polynomial time m-degree.

write $A \leqslant_1 B$. Note that if B is c.e. and $A \leqslant_m B$ then A is also c.e. Also note that $\emptyset'$ is *m-complete*, and even 1-*complete*, in the following sense.

Proposition 2.4.5. *If A is c.e. then $A \leqslant_1 \emptyset'$.*

Proof. It is easy to define an injective computable function f such that for each n, the machine $\Phi_{f(n)}$ ignores its input and halts iff n enters A at some point. Then f witnesses the fact that $A \leqslant_1 \emptyset'$. $\square$

It is not difficult to construct sets A and B such that $A \leqslant_T B$ but $A \nleqslant_m B$. For example, $\overline{\emptyset'} \leqslant_T \emptyset'$, but $\overline{\emptyset'} \nleqslant_m \emptyset'$, since $\overline{\emptyset'}$ is not c.e. It is also possible to construct such examples in which A and B are both c.e. Thus m-reducibility strictly refines Turing reducibility, and hence we say that m-reducibility is an example of a strong reducibility (which should not be confused with the notion of strong reducibility of mass problems introduced in Section 8.9). There are many other strong reducibilities. Their definitions depend on the types of oracle access used in the corresponding reductions. We mention two that will be important in this book.

One of the key aspects of Turing reducibility is that a Turing reduction may be adaptable, in the sense that the number and type of queries made of the oracle depends upon the oracle itself. For instance, imagine a reduction that works as follows: on input x, the oracle is queried as to whether it contains some power of x. That is, the reduction asks whether 1 is in the oracle, then whether x is in the oracle, then whether x^2 is in the oracle, and so on. If the answer is yes for some x^n, then the reduction checks whether the least such n is even, in which case it outputs 0, or odd, in which case it outputs 1.

Note that there is *no limit* to the number of questions asked of the oracle on a given input. This number depends on the oracle. Indeed, if the oracle happens to contain no power of x, then the computation on input x will not halt at all, and infinitely many questions will be asked.

Many naturally arising reductions do not have this adaptive property. One class of examples gives rise to the notion of *truth table reducibility.* A *truth table* on the variables $v_1, v_2, \ldots$ is a (finite) boolean combination σ of these variables. We write $A \vDash \sigma$ if σ holds with v_n interpreted as $n \in A$. For example, σ might be $((v_1 \wedge v_2) \vee (v_3 \rightarrow v_4)) \wedge v_5)$, in which case $A \vDash \sigma$ iff $5 \in A$ and either ($1 \in A$ and $2 \in A$) or ($3 \notin A$ or $4 \in A$) (or both). Let $\sigma_0, \sigma_1, \ldots$ be an effective list of all truth tables.

Definition 2.4.6. We say that A is *truth table reducible* to B, and write $A \leqslant_{tt} B$, if there is a computable function f such that for all x,

$$x \in A \text{ iff } B \vDash \sigma_{f(x)}.$$

Notice that an m-reduction is a simple example of a tt-reduction. The relevant truth table for input n has a single entry $v_{f(n)}$, where f is the function witnessing the given m-reduction.

The following characterization follows easily from the compactness of 2^ω.

Proposition 2.4.7 (Nerode [293]). *A Turing reduction Φ is a truth table reduction iff Φ^X is total for all oracles X.*

This characterization is particularly useful in the context of effective measure theory, as it means that truth table reductions can be used to transfer measures from one space to another. Examples can be found in the work of Reimann and Slaman [327, 328, 329] and the proof of Demuth's Theorem 8.6.1 below.

Concepts defined using Turing reducibility often have productive analogs for strong reducibilities. The following is an example we will use below.

Definition 2.4.8. A set A is *superlow* if $A' \equiv_{\mathrm{tt}} \emptyset'$.

One way to look at a tt-reduction is that it is one in which the oracle queries to be performed on a given input are predetermined, independently of the oracle, and the computation halts for every oracle. Removing the last restriction yields the notion of *weak truth table reduction*, which can also be thought of as bounded Turing reduction, in the sense that there is a computable bound, independent of the oracle, on the amount of the oracle to be queried on a given input.

Definition 2.4.9. We say that A is *weak truth table reducible* (*wtt-reducible*) to B, and write $A \leqslant_{\mathrm{wtt}} B$, if there are a computable function f and an oracle Turing machine Φ such that $\Phi^B = A$ and $\varphi^B(n) \leqslant f(n)$ for all n.

Definitions and notations that we introduced for Turing reducibility, such as the notion of degree, also apply to these strong reducibilities. In particular, we have the notions of *truth table functional* and *weak truth table functional*. For a Turing functional Φ to be a wtt-functional, we require that there be a single computable function f such that for all oracles B, if $\Phi^B(n)\downarrow$ then $\varphi^B(n) \leqslant f(n)$. The best way to think of a tt-functional is as a total Turing functional, that is, a functional Φ such that Φ^B is total for all oracles B.

The reducibilities we have seen so far calibrate sets into degrees of greater and greater fineness, in the order T, wtt, tt, m, 1.

2.4.4 Myhill's Theorem

The definition of $\emptyset'$ depends on the choice of an effective enumeration of the partial computable functions. By Proposition 2.4.5, however, any two versions of $\emptyset'$ are 1-equivalent. The following result shows that they are in fact equivalent up to a computable permutation of $\mathbb{N}$.

Theorem 2.4.10 (Myhill Isomorphism Theorem [291]). *$A \equiv_1 B$ iff there is a computable permutation h of $\mathbb{N}$ such that $h(A) = B$.*

Proof. The "if" direction is immediate, so assume that $A \leqslant_1 B$ via f and $B \leqslant_1 A$ via g. We define h in stages. We will ensure that the finite partial function h_s defined at each stage is injective and such that $m \in A$ iff $h_s(m) \in B$ for all $m \in \operatorname{dom} h_s$.

Let $h_0 = \emptyset$. Suppose we have defined h_s for s even. Let m be least such that $h_s(m)$ is not defined. List $f(n), f \circ h_s^{-1} \circ f(n), f \circ h_s^{-1} \circ f \circ h_s^{-1} \circ f(n), \dots$ until an element $k \notin \operatorname{rng} h_s$ is found. Note that each element of this list, and in particular k, is in B iff $n \in A$. Extend h_s to h_{s+1} by letting $h_{s+1}(m) = k$.

Now define h_{s+2} in the same way, with f, h_s, dom, and rng replaced by g, h_{s+1}^{-1}, rng, and dom, respectively.

Let $h = \bigcup_s h_s$. It is easy to check that h is a computable permutation of $\mathbb{N}$ and that $h(A) = B$. □

Myhill [291] characterized the 1-complete sets using the following notions.

Definition 2.4.11 (Post [316]). A set B is *productive* if there is a partial computable function h such that if $W_e \subseteq B$ then $h(e)\downarrow \in B \setminus W_e$.

A c.e. set A is *creative* if $\overline{A}$ is productive.

The halting problem is an example of a creative set, since $\overline{\emptyset'}$ is productive via the identity function.

Theorem 2.4.12 (Myhill's Theorem [291]). *The following are equivalent for a set A.*

(i) *A is creative.*

(ii) *A is m-complete (i.e., m-equivalent to $\emptyset'$).*

(iii) *A is 1-complete (i.e., 1-equivalent to $\emptyset'$).*

(iv) *A is equivalent to $\emptyset'$ up to a computable permutation of $\mathbb{N}$.*

Proof. The Myhill Isomorphism Theorem shows that (iii) and (iv) are equivalent, and (iii) obviously implies (ii). To show that (ii) implies (i), suppose that $\emptyset' \leqslant_m A$ via f. Let g be a computable function such that $W_{g(e)} = \{n : f(n) \in W_e\}$ for all e, and let $h(e) = f(g(e))$. If $W_e \subseteq \overline{A}$ then $W_{g(e)} \in \overline{\emptyset'}$, so $g(e) \in \overline{\emptyset'} \setminus W_{g(e)}$, and hence $h(e) \in \overline{A} \setminus W_e$. Thus A is creative via h.

To show that (i) implies (iii), we use the recursion theorem with parameters. Since A is c.e., we have $A \leqslant_1 \emptyset'$, so it is enough to show that $\emptyset' \leqslant_1 A$. Let h be a function witnessing that $\overline{A}$ is productive. Let f be a computable function such that $W_{f(n,k)} = \{f(n)\}$ if $k \in \emptyset'$ and $W_{f(n,k)} = \emptyset$ otherwise. By the recursion theorem with parameters, there is an injective computable function g such that $W_{g(k)} = W_{f(g(k),k)}$ for all k. Then

$$k \in \emptyset' \;\Rightarrow\; W_{g(k)} = \{f(g(k))\} \;\Rightarrow\; W_{g(k)} \nsubseteq \overline{A} \;\Rightarrow\; f(g(k)) \in A$$

and

$$k \notin \emptyset' \;\Rightarrow\; W_{g(k)} = \emptyset \;\Rightarrow\; W_{g(k)} \subseteq \overline{A} \;\Rightarrow\; f(g(k)) \in \overline{A}.$$

Thus $\emptyset' \leqslant_1 A$ via $f \circ g$. □

We will see a randomness-theoretic version of this result in Section 9.2.

2.5 The arithmetic hierarchy

We define the notions of Σ^0_n, Π^0_n, and Δ^0_n sets as follows. A set A is Σ^0_n if there is a computable relation $R(x_1, \ldots, x_n, y)$ such that $y \in A$ iff

$$\underbrace{\exists x_1\, \forall x_2\, \exists x_3\, \forall x_4 \cdots Q_n x_n}_{n \text{ alternating quantifiers}} R(x_1, \ldots, x_n, y).$$

Since the quantifiers alternate, Q_n is $\exists$ if n is odd and $\forall$ if n is even. In this definition, we could have had n alternating quantifier *blocks*, instead of single quantifiers, but we can always collapse two successive existential or universal quantifiers into a single one by using pairing functions, so that would not make a difference.

The definition of A being Π^0_n is the same, except that the leading quantifier is a $\forall$ (but there still are n alternating quantifiers in total). It is easy to see that A is Π^0_n iff $\overline{A}$ is Σ^0_n.

Finally, we say a set is Δ^0_n if it is both Σ^0_n and Π^0_n (or equivalently, if both it and its complement are Σ^0_n). Note that the Δ^0_0, Π^0_0, and Σ^0_0 sets are all exactly the computable sets. The same is true of the Δ^0_1 sets, as shown by Proposition 2.5.1 below.

These notions give rise to Kleene's *arithmetic hierarchy*, which can be pictured as follows.

$$\begin{tikzcd}
 & \Pi^0_1 \arrow[dr, hook, "/" marking] & & \Pi^0_2 \arrow[dr, hook, "/" marking] & & \\
\Delta^0_1 \arrow[ur, hook, "/" marking] \arrow[dr, hook, "/" marking] & & \Delta^0_2 \arrow[ur, hook, "/" marking] \arrow[dr, hook, "/" marking] & & \Delta^0_3 \arrow[ur, hook, "/" marking] \arrow[dr, hook, "/" marking] & \cdots \\
 & \Sigma^0_1 \arrow[ur, hook, "/" marking] & & \Sigma^0_2 \arrow[ur, hook, "/" marking] & &
\end{tikzcd}$$

A set is *arithmetic* if it is in one of the levels of the arithmetic hierarchy.

As we will see in the next section, there is a strong relationship between the arithmetic hierarchy and enumeration. The following is a simple example at the lowest level of the hierarchy

Proposition 2.5.1 (Kleene [208]). *A set A is computably enumerable iff A is Σ^0_1.*

Proof. Suppose A is c.e. Then $A = \operatorname{dom}(\Phi_e)$ for some e, so $n \in A$ iff $\exists s\, \Phi_e(y)[s]\!\downarrow$. Thus A is Σ^0_1.

Conversely, if A is Σ^0_1 then for some computable R we have $n \in A$ iff $\exists x\, R(x, n)$. Define a partial computable function g by letting $g(n) = 1$ at stage s iff $s \geqslant n$ and there is an $x < s$ such that $R(x, n)$ holds. Then $n \in A$ iff $n \in \operatorname{dom}(g)$, so A is c.e. $\square$

2.6 The Limit Lemma and Post's Theorem

There is an important generalization, due to Post, of Proposition 2.5.1. It ties in the arithmetic hierarchy with the degrees of unsolvability, and gives completeness properties of degrees of the form $\mathbf{0}^{(n)}$, highlighting their importance. In this section we will look at this and related characterizations, beginning with Shoenfield's Limit Lemma.

Saying that a set A is c.e. can be thought of as saying that A has a computable approximation that, for each n, starts out by saying that $n \notin A$, and then changes its mind at most once. More precisely, there is a computable binary function f such that for all n we have $A(n) = \lim_s f(n,s)$, with $f(n,0) = 0$ and $f(n,s+1) \neq f(n,s)$ for at most one s. Generalizing this idea, the limit lemma characterizes the sets computable from the halting problem as those that have computable approximations with *finitely many* mind changes, and hence are "effectively approximable". (In other words, the sets computable from $\emptyset'$ are exactly those that have computable approximations, as defined in Section 2.2.)

Theorem 2.6.1 (Limit Lemma, Shoenfield [354]). *For a set A, we have $A \leqslant_{\mathrm{T}} \emptyset'$ iff there is a computable binary function g such that, for all n,*

(i) $\lim_s g(n,s)$ *exists (i.e., $|\{s : g(n,s) \neq g(n,s+1)\}| < \infty$), and*

(ii) $A(n) = \lim_s g(n,s)$.

Proof. ($\Rightarrow$) Suppose $A = \Phi^{\emptyset'}$. Define g by letting $g(n,s) = 0$ if either $\Phi^{\emptyset'}[s]\uparrow$ or $\Phi^{\emptyset'}[s]\downarrow \neq 1$, and letting $g(n,s) = 1$ otherwise. Fix n, and let s be a stage such that $\emptyset'_t \upharpoonright \varphi^{\emptyset'}(n) = \emptyset' \upharpoonright \varphi^{\emptyset'}(n)$ for all $t \geqslant s$. By the use principle (Proposition 2.4.1), $g(n,t) = \Phi^{\emptyset'}(n)[t] = \Phi^{\emptyset'}(n) = A(n)$ for all $t \geqslant s$. Thus g has the required properties.

($\Leftarrow$) Suppose such a function g exists. Without loss of generality, we may assume that $g(n,0) = 0$ for all n. To show that $A \leqslant_{\mathrm{T}} \emptyset'$, it is enough to build a c.e. set B such that $A \leqslant_{\mathrm{T}} B$, since by Proposition 2.4.5, every c.e. set is computable in $\emptyset'$. We put $\langle n,k\rangle$ into B whenever we find that

$$|\{s : g(n,s) \neq g(n,s+1)\}| \geqslant k.$$

Now define a Turing reduction Γ as follows. Given an oracle X, on input n, search for the least k such that $\langle n,k\rangle \notin X$, and if one is found, then output 0 if k is even and 1 if k is odd. Clearly, $\Gamma^B = A$. $\square$

As in the case of c.e. sets, whenever we are given a set A computable in $\emptyset'$, we assume that we have a fixed computable approximation $A_0, A_1, \ldots$ to A; that is, we assume we have a computable function g as in the limit lemma, and write A_s for the set of all n such that $g(n,s) = 1$. We may always assume without loss of generality that $A_0 = \emptyset$ and $A_s \subseteq [0,s]$.

Intuitively, the proof of the "if" direction of the limit lemma boils down to saying that, since (by Propositions 2.4.5 and 2.5.1) the set $\emptyset'$ can decide whether $\exists s > t\,(g(n,s) \neq g(n,s+1))$ for any n and t, it can also compute $\lim_s g(n,s)$.

Corollary 2.6.2. *For a set A, the following are equivalent.*

(i) $A \leqslant_{\mathrm{tt}} \emptyset'$.

(ii) $A \leqslant_{\mathrm{wtt}} \emptyset'$.

(iii) *There are a computable binary function g and a computable function h such that, for all n,*

   (a) $|\{s : g(n,s) \neq g(n,s+1)\}| < h(n)$, *and*
   (b) $A(n) = \lim_s g(n,s)$.

Proof. We know that (i) implies (ii). The proof that (ii) implies (iii) is essentially the same as that of the "if" direction of Theorem 2.6.1, together with the remark that if Φ is a wtt-reduction then we can computably bound the number of times the value of $\Phi^{\emptyset'}[s]$ can change as s increases. The proof that (iii) implies (i) is much the same as that of the "only if" direction of Theorem 2.6.1, except that we can now make Γ into a tt-reduction because we have to check whether $\langle n,k\rangle \notin X$ only for $k < h(n)$. □

Sets with the properties given in Corollary 2.6.2 are called *ω-c.e.*, and will be further discussed in Section 2.7.

As we have seen, we often want to relativize results, definitions, and proofs in computability theory. The limit lemma relativizes to show that $A \leqslant_{\mathrm{T}} B'$ iff there is a B-computable binary function f such that $A(n) = \lim_s f(n,s)$ for all n. Combining this fact with induction, we have the following generalization of the limit lemma.

Corollary 2.6.3 (Limit Lemma, Strong Form, Shoenfield [354])**.** *Let $k \geqslant 1$. For a set A, we have $A \leqslant_{\mathrm{T}} \emptyset^{(k)}$ iff there is a computable function g of $k+1$ variables such that $A(n) = \lim_{s_1}\lim_{s_2}\dots\lim_{s_k} g(n,s_1,s_2,\dots,s_k)$ for all n.*

We now turn to Post's characterization of the levels of the arithmetic hierarchy. Let $\mathcal{C}$ be a class of sets (such as a level of the arithmetic hierarchy). A set A is *$\mathcal{C}$-complete* if $A \in \mathcal{C}$ and $B \leqslant_{\mathrm{T}} A$ for all $B \in \mathcal{C}$. If in fact $B \leqslant_{\mathrm{m}} A$ for all $B \in \mathcal{C}$, then we say that A is *$\mathcal{C}$ m-complete*, and similarly for other strong reducibilities.

Theorem 2.6.4 (Post's Theorem [317])**.** *Let $n \geqslant 0$.*

(i) *A set B is Σ^0_{n+1} $\Leftrightarrow$ B is c.e. in some Σ^0_n set $\Leftrightarrow$ B is c.e. in some Π^0_n set.*

(ii) *The set $\emptyset^{(n)}$ is Σ^0_n m-complete.*

(iii) *A set B is Σ^0_{n+1} iff B is c.e. in $\emptyset^{(n)}$.*

(iv) *A set B is Δ^0_{n+1} iff $B \leqslant_{\mathrm{T}} \emptyset^{(n)}$.*

Proof. (i) First note that if B is c.e. in A then B is also c.e. in $\overline{A}$. Thus, being c.e. in a Σ^0_n set is the same as being c.e. in a Π^0_n set, so all we need to show is that B is Σ^0_{n+1} iff B is c.e. in some Π^0_n set.

The "only if" direction has the same proof as the corresponding part of Proposition 2.5.1, except that the computable relation R in that proof is now replaced by a Π^0_n relation R.

For the "if" direction, let B be c.e. in some Π^0_n set A. Then, by Proposition 2.5.1 relativized to A, there is an e such that $n \in B$ iff

$$\exists s\, \exists \sigma \prec A\, (\Phi^\sigma_e(n)[s]\downarrow). \tag{2.1}$$

The property in parentheses is computable, while the property $\sigma \prec A$ is a combination of a Π^0_n statement (asserting that certain elements are in A) and a Σ^0_n statement (asserting that certain elements are not in A), and hence is Δ^0_{n+1}. So (2.1) is a Σ^0_{n+1} statement.

(ii) We proceed by induction. By Propositions 2.4.5 and 2.5.1, $\emptyset'$ is Σ^0_1 m-complete. Now assume by induction that $\emptyset^{(n)}$ is Σ^0_n m-complete. Since $\emptyset^{(n+1)}$ is c.e. in $\emptyset^{(n)}$, it is Σ^0_{n+1}. Let C be Σ^0_{n+1}. By part (i), C is c.e. in some Σ^0_n set, and hence it is c.e. in $\emptyset^{(n)}$. As in the unrelativized case, it is now easy to define a computable function f such that $n \in C$ iff $f(n) \in \emptyset^{(n+1)}$. (In more detail, let e be such that $C = W^{\emptyset^{(n)}}_e$, and define f so that for all oracles X and all n and x, we have $\Phi^X_{f(n)}(x)\downarrow$ iff $n \in W^X_e$.)

(iii) By (i) and (ii), and the fact that if X is c.e. in Y and $Y \leqslant_{\mathrm{T}} Z$, then X is also c.e. in Z.

(iv) The set B is Δ^0_{n+1} iff B and $\overline{B}$ are both Σ^0_{n+1}, and hence both c.e. in $\emptyset^{(n)}$ by (ii). But a set and its complement are both c.e. in X iff the set is computable in X. Thus B is Δ^0_{n+1} iff $B \leqslant_{\mathrm{T}} \emptyset^{(n)}$. ☐

Note in particular that the Δ^0_2 sets are exactly the $\emptyset'$-computable sets, that is, the sets that have computable approximations.

There are many "natural" sets, such as certain index sets, that are complete for various levels of the arithmetic hierarchy. The following result gives a few examples.

Theorem 2.6.5. (i) *Fin* $= \{e : W_e$ *is finite*$\}$ *is* Σ^0_2 *m-complete.*

(ii) *Tot* $= \{e : \Phi_e$ *is total*$\}$ *and Inf* $= \{e : W_e$ *is infinite*$\}$ *are both* Π^0_2 *m-complete.*

(iii) *Cof* $= \{e : W_e$ *is cofinite*$\}$ *is* Σ^0_3 *m-complete.*

Proof sketch. None of these are terribly difficult. We do (i) as an example. We know that $\emptyset''$ is Σ^0_2 m-complete by Post's Theorem, and it is easy to check that Fin is itself Σ^0_2, so it is enough to m-reduce $\emptyset''$ to Fin. Using the *s-m-n* theorem, we can define a computable function f such that for all s and e, we have $s \in W_{f(e)}$ iff there is a $t \geqslant s$ such that either $\Phi^{\emptyset'}_e(e)[t]\uparrow$ or $\emptyset'_{t+1} \upharpoonright \varphi^{\emptyset'}_e(e)[t] \neq \emptyset'_t \upharpoonright \varphi^{\emptyset'}_e(e)[t]$. Then $f(e) \in$ Fin iff $\Phi^{\emptyset'}_e(e)\downarrow$ iff $e \in \emptyset''$.

Part (ii) is similar, and (iii) is also similar but more intricate. See Soare [366] for more details. □

2.7 The difference hierarchy

The arithmetic hierarchy gives us one way to extend the concept of computable enumerability. Another way to do so is via the *difference hierarchy*, which is defined as follows.

Definition 2.7.1. Let $n \geqslant 1$. A set A is *n-computably enumerable* (*n-c.e.*) if there is a computable binary function f such that for all x,

(i) $f(x,0)=0$,

(ii) $A(x)=\lim_s f(x,s)$, and

(iii) $|\{s : f(x,s+1) \neq f(x,s)\}| \leqslant n$.

Thus the 1-c.e. sets are simply the c.e. sets. The 2-c.e. sets are often called *d.c.e.*, which stands for "difference of c.e.", because of the easily proved fact that A is 2-c.e. iff there are c.e. sets B and C such that $A = B \setminus C$.

We have seen the following definition in Section 2.6.

Definition 2.7.2. A set A is *ω-c.e.* if there are a computable binary function f and a computable unary function g such that for all x,

(i) $f(x,0)=0$,

(ii) $A(x)=\lim_s f(x,s)$, and

(iii) $|\{s : f(x,s+1) \neq f(x,s)\}| \leqslant g(x)$.

In Corollary 2.6.2, we saw that the ω-c.e. sets are exactly those that are (w)tt-reducible to $\emptyset'$. The following fact was probably known before it was explicitly stated by Arslanov [14].

Proposition 2.7.3 (Arslanov [14]). *If A is ω-c.e. then there is a $B \equiv_{\mathrm{m}} A$ and a computable binary function h such that for all x,*

(i) *$h(x,0)=0$,*

(ii) *$B(x)=\lim_s h(x,s)$, and*

(iii) *$|\{s : h(x,s+1) \neq h(x,s)\}| \leqslant x$.*

Proof. Let g be as in Definition 2.7.2. Without loss of generality, we may assume that g is increasing. Let $B = \{g(x) : x \in A\}$. Then B clearly has the desired properties. □

It is possible to define the concept of α-c.e. set for all computable ordinals α, forming what is known as the *Ershov hierarchy*. The details of the definition depend on ordinal notations. We will not use this concept in any significant way, so for simplicity we assume familiarity with the definition of a computable ordinal and Kleene's ordinal notations. For those unfamiliar with these concepts, we refer to Rogers [334] and Odifreddi [310, 311]. See Epstein, Haas, and Kramer [140] for more on α-c.e. sets and degrees.

Definition 2.7.4. Let α be a computable ordinal. A set A is *α-c.e.* relative to a computable system $\mathcal{S}$ of notations for α if there is a partial computable function f such that for all x, we have $A(x) = f(x, b)$ for the $\mathcal{S}$-least notation b such that $f(x, b)$ converges.

It is not hard to check that this definition agrees with the previous ones for $\alpha \leqslant \omega$, independently of the system of notations chosen.

As in the c.e. case, we say that a degree is n-c.e. if it contains an n-c.e. set, and similarly for ω-c.e. and α-c.e. degrees.

2.8 Primitive recursive functions

The class of *primitive recursive* functions is the smallest class of functions satisfying the following properties (where a 0-ary function is just a natural number).

(i) The function $n \mapsto n+1$ is primitive recursive.

(ii) For each k and m, the function $(n_0, \ldots, n_{k-1}) \mapsto m$ is primitive recursive.

(iii) For each k and $i < k$, the function $(n_0, \ldots, n_{k-1}) \mapsto n_i$ is primitive recursive.

(iv) If f and $g_0, \ldots, g_{k-1}$ are primitive recursive, f is k-ary, and the g_i are all j-ary, then

$$(n_0, \ldots, n_{j-1}) \mapsto f(g_0(n_0, \ldots, n_{j-1}), \ldots, g_{k-1}(n_0, \ldots, n_{j-1}))$$

is primitive recursive.

(v) If the k-ary function g and the $(k+2)$-ary function h are primitive recursive, then so is the function f defined by

$$f(0, n_0, \ldots, n_{k-1}) = g(n_0, \ldots, n_{k-1})$$

and

$$f(i+1, n_0, \ldots, n_{k-1}) = h(n, f(i, n_0, \ldots, n_{k-1}), n_0, \ldots, n_{k-1}).$$

It is easy to see that every primitive recursive function is computable. However, it is also easy to see that the primitive recursive functions are total and can be effectively listed, so there are computable functions that are not primitive recursive.[3] A famous example is *Ackermann's function*

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \wedge n = 0 \\ A(m-1, A(m,n-1)) & \text{otherwise.} \end{cases}$$

Many natural computable functions are primitive recursive, though, and it is sometimes useful to work with an effectively listable class of total computable functions, so we will use primitive recursive functions in a few places below.

2.9 A note on reductions

There are several ways to describe a reduction procedure. Formally, $A \leqslant_{\mathrm{T}} B$ means that there is an e such that $\Phi_e^B = A$. In practice, though, we never actually build an oracle Turing machine to witness the fact that $A \leqslant_{\mathrm{T}} B$, but avail ourselves of the Church-Turing Thesis to informally describe a reduction Γ such that $\Gamma^B = A$. One such description is given in the proof of the $\Leftarrow$ direction of Shoenfield's Limit Lemma (Theorem 2.6.1). This proof gives an example of a *static* definition of a reduction procedure, in that the action of Γ is specified by a rule, rather than being defined during a construction. As an example of a *dynamic* definition of a reduction procedure, we reprove the $\Leftarrow$ direction of the limit lemma.

Recall that we are given a computable binary function g such that, for all n,

(i) $\lim_s g(n,s)$ exists and

(ii) $A(n) = \lim_s g(n,s)$.

We wish to show that $A \leqslant_{\mathrm{T}} \emptyset'$, by building a c.e. set B and a reduction $\Gamma^B = A$.

We simultaneously construct B and Γ in stages. We begin with $B_0 = \emptyset$. For each n, we leave the value of $\Gamma^B(n)$ undefined until stage n. At stage n, we let $\Gamma^B(n)[n] = g(n,n)$ with use $\gamma^B(n)[n] = \langle n, 0\rangle + 1$.

Furthermore, at stage $s > 0$ we proceed as follows for each $n < s$. If $g(n,s) = g(n,s-1)$, then we change nothing. That is, we let $\Gamma^B(n)[s] = \Gamma^B(n)[s-1]$ with the same use $\gamma^B(n)[s] = \gamma^B(n)[s-1]$. Otherwise, we enumerate $\gamma(n,s-1) - 1$ into B, which allows us to redefine $\Gamma^B(n)[s] = g(n,s)$, with use $\gamma^B(n)[s] = \langle n,k\rangle + 1$ for the least $\langle n,k\rangle \notin B$.

It is not hard to check that $\Gamma^B = A$, and that in fact this reduction is basically the same as that in the original proof of the limit lemma (if we

[3]We can obtain the partial computable functions by adding to the five items above an unbounded search scheme. See Soare [366] for more details and further discussion of primitive recursive functions.

assume without loss of generality that $g(n,n)=0$ for all n), at least as far as its action on oracle B goes.

More generally, the rules for a reduction Δ^C to a c.e. set C are as follows, for each input n.

1. Initially $\Delta^C(n)[0]\uparrow$.

2. At some stage s we must define $\Delta^C(n)[s]\downarrow= i$ for some value i, with some use $\delta^C(n)[s]$. By this action, we are promising that $\Delta^C(n)=i$ unless $C\upharpoonright\delta^C(n)[s]\neq C_s\upharpoonright\delta^C(n)[s]$.

3. The convention now is that $\delta^C(n)[t]=\delta^C(n)[s]$ for $t>s$ unless $C_t\upharpoonright\delta^C(n)[s]\neq C_s\upharpoonright\delta^C(n)[t]$. Should we find a stage $t>s$ such that $C_t\upharpoonright\delta^C(n)[s]\neq C_s\upharpoonright\delta^C(n)[t]$, we then again have $\Delta^C(n)[t]\uparrow$.

4. We now again must have a stage $u\geqslant t$ at which we define $\Delta^C(n)[u]\downarrow=j$ for some value j, with some use $\delta^C(n)[u]$. We then return to step 3, with u in place of s.

5. If Δ^C is to be total, we have to ensure that we stay at step 3 permanently from some point on. That is, there must be a stage u at which we define $\Delta^C(n)[u]$ and $\delta^C(n)[u]$, such that $C\upharpoonright\delta^C(n)[u]=C_u\upharpoonright\delta^C(n)[u]$. One way to achieve this is to ensure that, from some point on, whenever we redefine $\delta^C(n)[u]$, we set it to the same value.

In some constructions, C will be given to us, but in others we will build it along with Δ. In this case, when we want to redefine the value of the computation $\Delta^C(n)$ at stage s, we will often be able to do so by putting a number less than $\delta^C(n)[s]$ into C (as we did in the limit lemma example above).

There is a similar method of building a reduction Δ^C when C is not c.e., but merely Δ^0_2. The difference is that now we must promise that if $\Delta^C(n)[s]$ is defined and there is a $t>s$ such that $C_t\upharpoonright\delta^C(n)[s]=C_s\upharpoonright\delta^C(n)[s]$, then $\Delta^C(n)[t]=\Delta^C(n)[s]$ and $\delta^C(n)[t]=\delta^C(n)[s]$.

A more formal view of a reduction is as a partial computable map from strings to strings obeying certain continuity conditions. In this view, a reduction $\Gamma^B=A$ is specified by a partial computable function $f:2^{<\omega}\to 2^{<\omega}$ such that

1. if $\sigma\prec B$, then $f(\sigma)\prec A$;

2. for all $\sigma\prec\tau$, if both $f(\sigma)\downarrow$ and $f(\tau)\downarrow$, then $f(\sigma)\preccurlyeq f(\tau)$; and

3. for all $\tau\prec A$ there is a $\sigma\prec B$ such that $\tau\prec f(\sigma)$.

The reduction in the proof of the $\Leftarrow$ direction of the limit lemma can be viewed in this way by letting $f(\sigma)$ be the longest string τ such that, for all $n<|\tau|$, there is a k with $\sigma(\langle n,k\rangle)=0$, and $\tau(n)=0$ iff the first such k to be found is even.

Notice that this last method implies the following interesting observation: *A function $f : 2^{<\omega} \to 2^{<\omega}$ is continuous iff it is computable relative to some oracle.*

2.10 The finite extension method

In this section, we introduce one of the main techniques used in classical degree theory. We will refine this technique in the next section to what is known as the finite injury priority method.

The dynamic construction of a reduction from B to A in the limit lemma, discussed in the previous section, has much in common with many proofs in classical computability theory. We perform a construction where some object is built in stages. Typically, we have some overall goal that we break down into smaller subgoals that we argue are all met in the limit. In this case, the goal is to construct the reduction $\Gamma^B = A$. We break this goal into the subgoals of defining $\Gamma^B(n)$ for each n, and we accomplish these subgoals by using the information supplied by our "opponent", who is feeding us information about the universe, in this case the values $g(n, s)$.

As an archetype for such proofs, think of Cantor's proof that the collection of all infinite binary sequences is uncountable. One can conceive of this proof as follows. Suppose we could list the infinite binary sequences as $\mathcal{S} = \{S_0, S_1, \dots\}$, with $S_e = s_{e,0}s_{e,1}\dots$. It is our goal to construct a binary sequence $U = u_0u_1\dots$ that is not on the list $\mathcal{S}$. We think of the construction as a game against our opponent who must supply us with $\mathcal{S}$. We construct u in stages, at stage t specifying only $u_0 \dots u_t$, the initial segment of U of length $t + 1$. Our list of *requirements* is the decomposition of the overall goal into subgoals of the form

$$\mathcal{R}_e : \; U \neq S_e.$$

There is one such requirement for each $e \in \mathbb{N}$. Of course, we know how to satisfy these requirements. At stage e, we simply ensure that $u_e \neq s_{e,e}$ by setting $u_e = 1 - s_{e,e}$. This action ensures that $U \neq S_e$ for all e; in other words, all the requirements are met. This fact contradicts the assumption that $\mathcal{S}$ lists all infinite binary sequences, as U is itself an infinite binary sequence.

Notice that if we define a real number to be *computable* if it has a computable binary expansion, then the above proof can be used to show that there is no computable listing of all the computable reals (modulo some unimportant technicalities involving nonunique representations of reals). We will return to the topic of effective real numbers in Chapter 5.

Clearly, the proof of the unsolvability of the halting problem can also be similarly recast, where this time the eth requirement asks us to invalidate the eth member of some supposed list of all algorithms deciding whether $\Phi_e(e)\downarrow$.

While later results will be more complicated than these easy examples, the overall structure of the above should be kept in mind: Our constructions will be in finite steps, where one or more objects are constructed stage by stage in finite pieces. These objects will be constructed to satisfy a list of requirements. The strategy we use will be dictated by how our opponent reveals the universe to us. Our overall goal is to satisfy all requirements in the limit.

To finish this section, we look at a slightly more involved version of this technique. While we know that there are uncountably many Turing degrees, the only ones we have seen so far are the iterates of the halting problem. Rice's Theorem 2.2.3 shows that all index sets are of degree $\geqslant \mathbf{0}'$. In 1944, Post [316] observed that all computably enumerable problems known at the time were either computable or of Turing degree $\mathbf{0}'$. He asked the following question.

Question 2.10.1 (Post's Problem)**.** Does there exist a computably enumerable degree $\mathbf{a}$ with $\mathbf{0} < \mathbf{a} < \mathbf{0}'$?

As we will see in the next section, Post's Problem was finally given a positive answer by Friedberg [161] and Muchnik [284], using a new and ingenious method called the priority method. This method was an effectivization of an earlier method discovered by Kleene and Post [210]. The latter is called the finite extension method, and was used to prove the following result.

Theorem 2.10.2 (Kleene and Post [210])**.** *There are degrees* $\mathbf{a}$ *and* $\mathbf{b}$*, both below* $\mathbf{0}'$*, such that* $\mathbf{a} \mid \mathbf{b}$*. In other words, there are* $\emptyset'$*-computable sets that are incomparable under Turing reducibility.*[4]

Proof. We construct $A = \lim_s A_s$ and $B = \lim_s B_s$ in stages, to meet the following requirements for all $e \in \mathbb{N}$.

$$\mathcal{R}_{2e} : \Phi_e^A \neq B.$$
$$\mathcal{R}_{2e+1} : \Phi_e^B \neq A.$$

Note that if $A \leqslant_{\mathrm{T}} B$ then there must be some procedure Φ_e with $\Phi_e^B = A$. Hence, if we meet all our requirements then $A \nleqslant_{\mathrm{T}} B$, and similarly $B \nleqslant_{\mathrm{T}} A$, so that A and B have incomparable Turing degrees. The fact that $A, B \leqslant_{\mathrm{T}} \emptyset'$ will come from the construction and will be observed at the end.

The argument is by finite extensions, in the sense that at each stage s we specify a finite portion A_s of A and a finite portion B_s of B. These finite portions A_s and B_s will be specified as binary strings. The key invariant that we need to maintain throughout the construction is that $A_s \preccurlyeq A_u$ and

[4]The difference between the Kleene-Post Theorem and the solution to Post's Problem is that the degrees constructed in the proof of Theorem 2.10.2 are not necessarily computably enumerable, but merely Δ_2^0.

$B_s \preccurlyeq B_u$ for all stages $u \geqslant s$. Thus, after stage s we can only *extend* the portions of A and B that we have specified by stage s, which is a hallmark of the finite extension method.

Construction.
Stage 0. Let $A_0 = B_0 = \lambda$ (the empty string).

Stage $2e+1$. (Attend to $\mathcal{R}_{2e}$.) We will have specified A_{2e} and B_{2e} at stage $2e$. Pick some number x, called a *witness*, with $x \geqslant |B_{2e}|$, and ask whether there is a string σ properly extending A_{2e} such that $\Phi_e^\sigma(x)\downarrow$.

If such a σ exists, then let A_{2e+1} be the length-lexicographically least such σ. Let B_{2e+1} be the string of length $x+1$ extending B_{2e} such that $B_{2e+1}(n) = 0$ for all n with $|B_{2e}| \leqslant n < x$ and $B_{2e+1}(x) = 1 - \Phi_e^\sigma(x)$.

If no such σ exists, then let $A_{2e+1} = A_{2e}0$ and $B_{2e+1} = B_{2e}0$.

Stage $2e+2$. (Attend to $\mathcal{R}_{2e+1}$.) Define A_{2e+2} and B_{2e+2} by proceeding in the same way as at stage $2e+1$, but with the roles of A and B reversed.
End of Construction.

Verification. First note that we have $A_0 \prec A_1 \prec \cdots$ and $B_0 \prec B_1 \prec \cdots$, so A and B are well-defined.

We now prove that we meet the requirement $\mathcal{R}_n$ for each n; in fact, we show that we meet $\mathcal{R}_n$ at stage $n+1$. Suppose that $n = 2e$ (the case where n is odd being completely analogous). At stage $n+1$, there are two cases to consider. Let x be as defined at that stage.

If there is a σ properly extending A_n with $\Phi_e^\sigma(x)\downarrow$, then our action is to adopt such a σ as A_{n+1} and define B_{n+1} so that $\Phi_e^{A_{n+1}}(x) \neq B_{n+1}(x)$. Since A extends A_{n+1} and $\Phi_e^{A_{n+1}}(x)\downarrow$, it follows that A and A_{n+1} agree on the use of this computation, and hence $\Phi_e^A(x) = \Phi_e^{A_{n+1}}$. Since B extends B_{n+1}, we also have $B(x) = B_{n+1}(x)$. Thus $\Phi_e^A(x) \neq B(x)$, and $\mathcal{R}_n$ is met.

If there is no σ extending A_n with $\Phi_e^\sigma(x)\downarrow$, then since A is an extension of A_n, it must be the case $\Phi^A(x)\uparrow$, and hence $\mathcal{R}_n$ is again met.

Finally we argue that $A, B \leqslant_{\mathrm{T}} \emptyset'$. Notice that the construction is in fact fully computable except for the decision as to which case we are in at a given stage. There we must decide whether there is a convergent computation of a particular kind. For instance, at stage $2e+1$ we must decide whether the following holds:

$$\exists\tau\,\exists s\,[\tau \succ A_{2e} \wedge \Phi_e^\tau(x)[s]\downarrow]. \tag{2.2}$$

This is a Σ_1^0 question, uniformly in x, and hence can be decided by $\emptyset'$.[5] □

The reasoning at the end of the above proof is quite common: we often make use of the fact that $\emptyset'$ can answer any Δ_2^0 question, and hence any Σ_1^0 or Π_1^0 question.

[5]More precisely, we use the s-m-n theorem to construct a computable ternary function f such that for all e, σ, x, and z, we have $\Phi_{f(e,\sigma,x)}(z)\downarrow$ iff (2.2) holds. Then (2.2) holds iff $f(e,\sigma,x) \in \emptyset'$.

A key ingredient of the proof of Theorem 2.10.2 is the use principle (Proposition 2.4.1). In constructions of this sort, where we build objects to defeat certain oracle computations, a typical requirement will say something like "the reduction Γ is not a witness to $A \leqslant_{\mathrm{T}} B$." If we have a converging computation $\Gamma^B(n)[s] \neq A(n)[s]$ and we "preserve the use" of this computation by not changing B after stage s on the use $\gamma^B(n)[s]$ (and similarly preserve $A(n)$), then we will preserve this disagreement. But this use corresponds to only a finite portion of B, so we still have all the numbers bigger than it to meet other requirements. In the finite extension method, this use preservation is automatic, since once we define $B(x)$ we never redefine it, but in other constructions we will introduce below, this may not be the case, because we may have occasion to redefine certain values of B. In that case, to ensure that $\Gamma^B \neq A$, we will have to structure the construction so that, if Γ^B is total, then there are n and s such that $\Gamma^B(n)[s] \neq A(n)[s]$ and, from stage s on, we preserve both $A(n)$ and $B \upharpoonright \gamma^B(n)[s]$.

2.11 Post's Problem and the finite injury priority method

A more subtle technique than the finite extension method is the *priority method*. We begin by looking at the simplest incarnation of this elegant technique, the *finite injury priority method*. This method is somewhat like the finite extension method, but with backtracking.

The idea behind it is the following. Suppose we must again satisfy requirements $\mathcal{R}_0, \mathcal{R}_1, \ldots$, but this time we are constrained to some sort of effective construction, so we are not allowed to ask questions of a noncomputable oracle during the construction. As an illustration, let us reconsider Post's Problem (Question 2.10.1). Post's Problem asks us to find a c.e. degree strictly between $\mathbf{0}$ and $\mathbf{0}'$. It is clearly enough to construct c.e. sets A and B with incomparable Turing degrees. The Kleene-Post method does allow us to construct sets with incomparable degrees below $\mathbf{0}'$, using a $\emptyset'$ oracle question at each stage, but there is no reason to expect these sets to be computably enumerable. To make A and B c.e., we must have a *computable* (rather than merely $\emptyset'$-computable) construction where elements go into the sets A and B but never leave them. As we will see, doing so requires giving up on satisfying our requirements in order. The key idea, discovered independently by Friedberg [161] and Muchnik [284], is to pursue multiple strategies for each requirement, in the following sense.

In the proof of the Kleene-Post Theorem, it appears that, in satisfying the requirement $\mathcal{R}_{2e}$, we need to know whether or not there is a σ extending A_{2e} such that $\Phi_e^\sigma(x)\downarrow$, where x is our chosen witness. Now our idea is to first *guess* that no such σ exists, which means that we do nothing for $\mathcal{R}_{2e}$

other than keep x out of B. If at some point we find an appropriate σ, we then make A extend σ and put x into B if necessary, as in the Kleene-Post construction.

The only problem is that putting x into B may well upset the action of other requirements of the form $\mathcal{R}_{2i+1}$, because such a requirement might need B to extend some string τ (for the same reason that $\mathcal{R}_{2e}$ needs A to extend σ), which may no longer be possible. If we nevertheless put x into B, we say that we have *injured* $\mathcal{R}_{2i+1}$. Of course, $\mathcal{R}_{2i+1}$ can now choose a new witness and start over from scratch, but perhaps another requirement may injure it again later. So we need to somehow ensure that, for each requirement, there is a stage after which it is never injured.

To make sure that this is the case, we put a *priority ordering* on our requirements, by stating that $\mathcal{R}_j$ has stronger priority than $\mathcal{R}_i$ if $j < i$, and allow $\mathcal{R}_j$ to injure $\mathcal{R}_i$ only if $\mathcal{R}_j$ has stronger priority than $\mathcal{R}_i$. Thus $\mathcal{R}_0$ is never injured. The requirement $\mathcal{R}_1$ may be injured by the action of $\mathcal{R}_0$. However, once this happens $\mathcal{R}_0$ will never act again, so if $\mathcal{R}_1$ is allowed to start over at this point, it will succeed. This process of starting over is called *initialization*. Initializing $\mathcal{R}_1$ means that we restart its action with a new witness, chosen to be larger than any number previously seen in the construction, and hence larger than any number $\mathcal{R}_0$ cares about. This new incarnation of $\mathcal{R}_1$ is guaranteed never to be injured. It should now be clear that, by induction, each requirement will eventually reach a point, following a finite number of initializations, after which it will never be injured and hence will succeed in reaching its goal.

We may think of this kind of construction as a game between a team of industrialists (each possibly trying to erect a factory) and a team of environmentalists (each possibly trying to build a park). In the end we want the world to be happy. In other words, we want all desired factories and parks to be built. However, some of the players may get distracted by other activities and never decide to build anything, so we cannot simply let one player build, then the next, and so on, because we might then get permanently stuck waiting for a player who never decides to build. Members of the two teams have their own places in the pecking order. For instance, industrialist 6 has stronger priority than all environmentalists except the first six, and therefore can build anywhere except on parks built by the first six environmentalists. So industrialist 6 may choose to build on land already demarcated by environmentalist 10, say, who would then need to find another place to build a park. Of course, even if this event happens, a higher ranked environmentalist, such as number 3, for instance, could later lay claim to that same land, forcing industrialist 6 to find another place to build a factory. Whether the highest ranked industrialist has priority over the highest ranked environmentalist or vice versa is irrelevant to the construction, so we leave that detail to each reader's political leanings.

For each player, there are only finitely many other players with stronger priority, and once all of these have finished building what they desire, the given player has free pick of the remaining land (which is infinite), and can build on it without later being forced out.

In general, in a finite injury priority argument, we have a list of requirements in some priority ordering. There are several different ways to meet each individual requirement. Exactly which way will be possible to implement depends upon information that is not initially available to us but is "revealed" to us during the construction. The problem is that a requirement cannot wait for others to act, and hence must risk having its work destroyed by the actions of other requirements. We must arrange things so that only requirements of stronger priority can injure ones of weaker priority, and we can always restart the ones of weaker priority once they are injured. In a finite injury argument, any requirement *requires attention* only finitely often, and we argue by induction that each requirement eventually gets an environment wherein it can be met. As we will later see, there are much more complex infinite injury arguments where one requirement might injure another infinitely often, but the key there is that the injury is somehow controlled so that it is still the case that each requirement eventually gets an environment wherein it can be met. Of course, imposing this coherence criterion on our constructions means that each requirement must ensure that its action does not prevent weaker requirements from finding appropriate environments (a principle known as Harrington's "golden rule").

For a more thorough account of these beautiful techniques and their uses in modern computability theory, see Soare [366].

We now turn to the formal description of the solution to Post's Problem by Friedberg and Muchnik, which was the first use of the priority method. In Chapter 11, we will return to Post's Problem and explore its connections with the notion of Kolmogorov complexity.

Theorem 2.11.1 (Friedberg [161], Muchnik [284]). *There exist computably enumerable sets A and B such that A and B have incomparable Turing degrees.*

Proof. We build $A = \bigcup_s A_s$ and $B = \bigcup_s B_s$ in stages to satisfy the same requirements as in the proof of the Kleene-Post Theorem. That is, we make A and B c.e. while meeting the following requirements for all $e \in \mathbb{N}$.

$$\mathcal{R}_{2e} : \Phi_e^A \neq B.$$
$$\mathcal{R}_{2e+1} : \Phi_e^B \neq A.$$

The strategy for a single requirement. We begin by looking at the strategy for a single requirement $\mathcal{R}_{2e}$. We first pick a witness x to *follow* $\mathcal{R}_{2e}$. This *follower* is targeted for B, and, of course, we initially keep it out of B. We then wait for a stage s such that $\Phi_e^A(x)[s]\downarrow = 0$. If such a stage does not occur, then either $\Phi_e^A(x)\uparrow$ or $\Phi_e^A(x)\downarrow \neq 0$. In either case, since we keep x out of B, we have $\Phi_e^A(x) \neq 0 = B(x)$, and hence $\mathcal{R}_{2e}$ is satisfied.

If a stage s as above occurs, then we put x into B and *protect* A_s. That is, we try to ensure that any number entering A from now on is greater than any number seen in the construction thus far, and hence in particular greater than $\varphi_e^A(x)[s]$. If we succeed then, by the use principle, $\Phi_e^A(x) = \Phi_e^A(x)[s] = 0 \neq B(x)$, and hence again $\mathcal{R}_{2e}$ is satisfied. We refer to this action of protecting A_s as imposing *restraint* on weaker priority requirements.

Note that when we take this action, we might injure a requirement $\mathcal{R}_{2i+1}$ that is trying to preserve the use of a computation $\Phi_i^B(x')$, since x may be below this use. As explained above, the priority mechanism will ensure that this can happen only if $2i + 1 > 2e$.

We now proceed with the full construction. We will denote by A_s and B_s the sets of elements enumerated into A and B, respectively, by the end of stage s.

Construction.
Stage 0. Declare that no requirement currently has a follower.

Stage $s + 1$. Say that $\mathcal{R}_j$ *requires attention* at this stage if one of the following holds.

(i) $\mathcal{R}_j$ currently has no follower.

(ii) $\mathcal{R}_j$ has a follower x and, for some e, either
  (a) $j = 2e$ and $\Phi_e^A(x)[s]\downarrow= 0 = B_s(x)$ or
  (b) $j = 2e + 1$ and $\Phi_e^B(x)[s]\downarrow= 0 = A_s(x)$.

Find the least $j \leqslant s$ with $\mathcal{R}_j$ requiring attention. (If there is none, then proceed to the next stage.) We suppose that $j = 2e$, the odd case being symmetric. If $\mathcal{R}_{2e}$ has no follower, then let x be a *fresh large* number (that is, one larger than all numbers seen in the construction so far) and appoint x as $\mathcal{R}_{2e}$'s follower.

If $\mathcal{R}_{2e}$ has a follower x, then it must be the case that $\Phi_e^A(x)[s]\downarrow= 0 = B_s(x)$. In this case, enumerate x into B and *initialize* all $\mathcal{R}_k$ with $k > 2e$ by canceling all their followers.

In either case, we say that $\mathcal{R}_{2e}$ *receives attention* at stage s.
End of Construction.

Verification. We prove by induction that, for each j,

(i) $\mathcal{R}_j$ receives attention only finitely often, and

(ii) $\mathcal{R}_j$ is met.

Suppose that (i) holds for each $k < j$ in place of j. Suppose that $j = 2e$ for some e, the odd case being symmetric. Let s be the least stage such that for all $k < j$, the requirement $\mathcal{R}_k$ does not require attention after stage s. By the minimality of s, some requirement $\mathcal{R}_k$ with $k < j$ received attention at stage s (or $s = 0$), and hence $\mathcal{R}_j$ does not have a follower at the beginning of stage $s+1$. Thus, $\mathcal{R}_j$ requires attention at stage $s+1$, and is appointed a follower x. Since $\mathcal{R}_j$ cannot have its follower canceled unless some $\mathcal{R}_k$ with $k < j$ receives attention, x is $\mathcal{R}_j$'s permanent follower.

It is clear by the way followers are chosen that x is never any other requirement's follower, so x will not enter B unless $\mathcal{R}_j$ acts to put it into B. So if $\mathcal{R}_j$ never requires attention after stage $s+1$, then $x \notin B$, and we never have $\Phi_e^A(x)[t] \downarrow= 0$ for $t > s$, which implies that either $\Phi_e^A(x) \uparrow$ or $\Phi_e^A(x) \downarrow\neq 0$. In either case, $\mathcal{R}_j$ is met.

On the other hand, if $\mathcal{R}_j$ requires attention at a stage $t+1 > s+1$, then $x \in B$ and $\Phi_e^A(x)[t] \downarrow= 0$. The only requirements that put numbers into A after stage $t+1$ are ones weaker than $\mathcal{R}_j$ (i.e., requirements $\mathcal{R}_k$ for $k > j$). Each such strategy is initialized at stage $t+1$, which means that, when it is later appointed a follower, that follower will be bigger than $\varphi_e^A(x)[t]$. Thus no number less than $\varphi_e^A(x)[t]$ will ever enter A after stage $t+1$, which implies, by the use principle, that $\Phi^A(x) \downarrow= \Phi_e^A(x)[t] = 0 \neq B(x)$. So in this case also, $\mathcal{R}_j$ is met. Since $x \in B_{t+2}$ and x is $\mathcal{R}_j$'s permanent follower, $\mathcal{R}_j$ never requires attention after stage $t+1$. □

The above proof is an example of the simplest kind of finite injury argument, what is called a *bounded injury* construction. That is, we can put a computable bound *in advance* on the number of times that a given requirement $\mathcal{R}_j$ will be injured. In this case, the bound is $2^j - 1$.

We give another example of this kind of construction, connected with the important concept of lowness. It is natural to ask what can be said about the jump operator beyond the basic facts we have seen so far. The next theorem proves that the jump operator on degrees is not injective. Indeed, injectivity fails in the first place it can, in the sense that there are noncomputable sets that the jump operator cannot distinguish from $\emptyset$.

Theorem 2.11.2 (Friedberg). *There is a noncomputable c.e. low set.*

Proof. We construct our set A in stages. To make A noncomputable we need to meet the requirements

$$\mathcal{P}_e : \overline{A} \neq W_e.$$

To make A low we meet the requirements

$$\mathcal{N}_e : (\exists^\infty s\, \Phi_e^A(e)[s]\downarrow) \Rightarrow \Phi_e^A(e)\downarrow .$$

To see that such requirements suffice, suppose they are met and define the computable binary function g by letting $g(e,s) = 1$ if $\Phi_e^A(e)[s]\downarrow$ and $g(e,s) = 0$ otherwise. Then $g(e) = \lim_s g(e,s)$ is well-defined, and by the limit lemma, $A' = \{e : g(e) = 1\} \leqslant_{\mathrm{T}} \emptyset'$.

The strategy for $\mathcal{P}_e$ is simple. We pick a fresh large follower x, and keep it out of A. If x enters W_e, then we put x into A. We meet $\mathcal{N}_e$ by an equally simple conservation strategy. If we see $\Phi_e^A(e)[s]\downarrow$ then we simply try to ensure that $A \upharpoonright \varphi_e^A(e)[s] = A_s \upharpoonright \varphi_e^A(e)[s]$ by initializing all weaker priority requirements, which forces them to choose fresh large numbers as followers. These numbers will be too big to injure the $\Phi_e^A(e)[s]$ computation after stage s. The priority method sorts the actions of the various strategies out. Since $\mathcal{P}_e$ picks a fresh large follower each time it is initialized, it cannot injure any $\mathcal{N}_j$ for $j < e$. It is easy to see that any $\mathcal{N}_e$ can be injured at most e many times, and that each $\mathcal{P}_e$ is met, since it is initialized at most 2^e many times. □

Actually, the above proof constructs a noncomputable c.e. set that is superlow. (Recall that a set A is superlow if $A' \equiv_{\mathrm{tt}} \emptyset'$.)

2.12 Finite injury arguments of unbounded type

2.12.1 The Sacks Splitting Theorem

There are priority arguments in which the number of injuries to each requirement, while finite, is not bounded by any computable function. One example is the following proof of the Sacks Splitting Theorem [342]. We write $A = A_0 \sqcup A_1$ to mean that $A = A_0 \cup A_1$ and $A_0 \cap A_1 = \emptyset$. Turing incomparable c.e. sets A_0 and A_1 such that $A = A_0 \sqcup A_1$ are said to form a *c.e. splitting* of A.

Theorem 2.12.1 (Sacks Splitting Theorem [342]). *Every noncomputable c.e. set has a c.e. splitting.*

Proof. Let A be a noncomputable c.e. set. We build $A_i = \bigcup_s A_{i,s}$ in stages by a priority argument to meet the following requirements for all $e \in \mathbb{N}$ and $i = 0, 1$, while ensuring that $A = A_0 \sqcup A_1$.

$$\mathcal{R}_{e,i} :\ \Phi_e^{A_i} \neq A.$$

These requirements suffice because if $A_{1-i} \leqslant_{\mathrm{T}} A_i$ then $A \leqslant_{\mathrm{T}} A_i$.

Without loss of generality, we assume that we are given an enumeration of A so that exactly one number enters A at each stage. We must put this number $x \in A_{s+1} \setminus A_s$ into exactly one of A_0 or A_1, to ensure that $A = A_0 \sqcup A_1$.

To meet $\mathcal{R}_{e,i}$, we define the *length of agreement function*

$$l(e,i,s) = \max\{n : \forall k < n\,(\Phi_e^{A_i}(k)[s] = A(k)[s])\}$$

and an associated use function

$$u(e,i,s) = \varphi_e^{A_i}(l(e,i,s)-1)[s],^{6}$$

using the convention that use functions are monotone increasing (with respect to the position variable) where defined.

[6]Of course, in defining this set we ignore s's such that $l(e,i,s) = 0$. We will do the same without further comment below. Here and below, we take the maximum of the empty set to be 0.

The main idea of the proof is perhaps initially counterintuitive. Let us consider a single requirement $\mathcal{R}_{e,i}$ in isolation. At each stage s, although we want $\Phi_e^{A_i} \neq A$, instead of trying to destroy the agreement between $\Phi_e^{A_i}[s]$ and $A[s]$ represented by $l(e,i,s)$, we try to *preserve* it (a method sometimes called the *Sacks preservation strategy*). The way we implement this preservation is to put numbers entering $A \upharpoonright u(e,i,s)$ after stage s into A_{1-i} and *not* into A_i. By the use principle, since this action freezes the A_i side of the computations involved in the definition of $l(e,i,s)$, it ensures that $\Phi_e^{A_i}(k) = \Phi_e^{A_i}(k)[s]$ for all $k < l(e,i,s)$.

Now suppose that $\limsup_s l(e,i,s) = \infty$, so for each k we can find infinitely many stages s at which $k < l(e,i,s)$. For each such stage, $\Phi_e^{A_i}(k) = \Phi_e^{A_i}(k)[s] = A(k)[s]$. Thus $A(k) = A(k)[s]$ for any such s. So we can compute $A(k)$ simply by finding such an s, which contradicts the noncomputability of A. Thus $\limsup_s l(e,i,s) < \infty$, which clearly implies that $\mathcal{R}_{e,i}$ is met.

In the full construction, of course, we have competing requirements, which we sort out by using priorities. That is, we establish a priority list of our requirements (for instance, saying that $\mathcal{R}_{e,i}$ is stronger than $\mathcal{R}_{e',i'}$ iff $\langle e,i\rangle < \langle e',i'\rangle$). At stage s, for the single element x_s entering A at stage s, we find the strongest priority $\mathcal{R}_{e,i}$ with $\langle e,i\rangle < s$ such that $x_s < u(e,i,s)$ and put x_s into A_{1-i}. We say that $\mathcal{R}_{e,i}$ *acts* at stage s. (If there is no such requirement, then we put x_s into A_0.)

To verify that this construction works, we argue by induction that each requirement eventually stops acting and is met. Suppose that all requirements stronger than $\mathcal{R}_{e,i}$ eventually stop acting, say by a stage $s > \langle e,i\rangle$. At any stage $t > s$, if $x_t < u(e,i,t)$, then x_t is put into A_{1-i}. The same argument as in the one requirement case now shows that if $\limsup_s l(e,i,s) = \infty$ then A is computable, so $\limsup_s l(e,i,s) < \infty$. Our preservation strategy then ensures that $\limsup_s u(e,i,s) < \infty$. Thus $\mathcal{R}_{e,i}$ eventually stops acting and is met. $\square$

In the above construction, injury to a requirement $\mathcal{R}_{e,i}$ happens whenever $x_s < u(e,i,s)$ but x_s is nonetheless put into A_i, at the behest of a stronger priority requirement. How often $\mathcal{R}_{e,i}$ is injured depends on the lengths of agreement attached to stronger priority requirements, and thus cannot be computably bounded.

Note that, for any noncomputable c.e. set C, we can easily add requirements of the form $\Phi_e^{A_i} \neq C$ to the above construction, satisfying them in the same way that we did for the $\mathcal{R}_{e,i}$. Thus, as shown by Sacks [342], in addition to making $A_0 \mid_{\mathrm{T}} A_1$, we can also ensure that $A_i \not\geqslant_{\mathrm{T}} C$ for $i = 0,1$.

Note also that the computable enumerability of A is not crucial in the above argument. Indeed, a similar argument works for any set A that has a computable approximation, that is, any Δ_2^0 set A. Such an argument shows that if A and C are noncomputable Δ_2^0 sets, then there exist Turing incomparable Δ_2^0 sets A_0 and A_1 such that $A = A_0 \sqcup A_1$ and $A_i \not\geqslant_{\mathrm{T}} C$

for $i = 0, 1$. Checking that the details of the proof still work in this case is a good exercise for those unfamiliar with priority arguments involving Δ^0_2 sets.

2.12.2 The Pseudo-Jump Theorem

Another basic construction using the finite injury method was discovered by Jockusch and Shore [193]. It involves what are called *pseudo-jump operators.*

Definition 2.12.2 (Jockusch and Shore [193]). For an index e, let the *pseudo-jump operator* V_e be defined by $V_e^A = A \oplus W_e^A$ for all oracles A. We say that V_e is *nontrivial* if $A <_{\mathrm{T}} V_e^A$ for all oracles A.

The jump operator is itself a nontrivial pseudo-jump operator (up to degree). The following pseudo-jump inversion theorem is an important tool in the theory of pseudo-jump operators.

Theorem 2.12.3 (Jockusch and Shore [193]). *For any nontrivial pseudo-jump operator V, there is a noncomputable c.e. set A with $V^A \equiv_{\mathrm{T}} \emptyset'$.*

Proof. We build $A = \bigcup_s A_s$ in stages. Let k be such that $V^X = X \oplus W_k^X$ for all X.

To ensure that $V^A \leqslant_{\mathrm{T}} \emptyset'$, we meet the requirements

$$\mathcal{N}_n : \exists^\infty s\,(n \in W_k^A[s]) \;\Rightarrow\; n \in W_k^A,$$

which imply that W_k^A is Π^0_2. Since W_k^A is necessarily Σ^0_2 and V^A is the join of W_k^A with a c.e. set, these requirements suffice to ensure that V^A is Δ^0_2. To satisfy $\mathcal{N}_n$, we have a *restraint function* $r(n,s)$, which we define as follows. If $n \in W_k^A[s]$, then $r(n,s)$ is the use of the computation ensuring this fact (that is, $\varphi_k^A(n)[s]$). Otherwise, $r(n,s) = 0$.

Let $\gamma(n,s)$ be the least element of $\mathbb{N}^{[n]}$ greater than or equal to $r(m,s)$ for all $m \leqslant n$. We use these numbers both to make A noncomputable and to ensure that $\emptyset' \leqslant_{\mathrm{T}} V^A$. We will make sure that $\gamma(n) = \lim_s \gamma(n,s)$ exists for all n.

To make A noncomputable, we meet the requirements

$$\mathcal{R}_e : \overline{W_e} \neq A,$$

by putting $\gamma(2e,s)$ into A if we have not yet satisfied $\mathcal{R}_e$ and $\gamma(2e,s) \in W_e[s]$.

To have $\emptyset' \leqslant_{\mathrm{T}} V^A$, we ensure that, for each n, we can V^A-computably determine a stage u such that $\gamma(n,s) = \gamma(n,u)$ for all $s \geqslant u$, and satisfy

$$\mathcal{P}_n : n \in \emptyset' \;\Leftrightarrow\; \exists s\,(\gamma(2n+1,s) \in A).$$

To satisfy $\mathcal{P}_n$, if n enters $\emptyset'$ at stage s, then we put $\gamma(2n+1,s)$ into A.

Thus the full construction proceeds as follows at stage s: For each e such that $W_e[s] \cap A[s] = \emptyset$ and $\gamma(2e, s) \in W_e[s]$, put $\gamma(2e, s)$ into A. For each n entering $\emptyset'$ at stage s, put $\gamma(2n+1, s)$ into A.

We put at most one number into A for the sake of a given $\mathcal{R}$- or $\mathcal{P}$-requirement. By the definition of the γ function, for each n there is an s such that, for all $t \geqslant s$, no number less than $r(n,t)$ is put into A at stage t. It follows that $\lim_t r(n,t)$ exists, and $\mathcal{N}_n$ is satisfied.

Thus $\gamma(n) = \lim_s \gamma(n,s)$ exists for all n, and the construction ensures that if $W_e \cap A = \emptyset$ then $\gamma(2e) \in A$ iff $\gamma(2e) \in W_e$, so that $\mathcal{R}_e$ is satisfied.

It is also clear that each $\mathcal{P}$-requirement is satisfied by the construction, so we are left with showing that for each n, we can V^A-computably determine a stage u such that $\gamma(n,s) = \gamma(n,u)$ for all $s \geqslant u$. Assume by induction that we have determined a stage t such that $\gamma(m,s) = \gamma(m,t)$ for all $m < n$ and $s \geqslant t$. Let $u > t$ be a stage such that for all $m \leqslant n$, if $m \in W_k^A$ then $m \in W_k^A[u]$. Clearly, such a u can be found V^A-computably. Let $m \leqslant n$. If $m \in W_k^A$ then $r(m,s) = r(m,u)$ for all $s \geqslant u$, since no number less than $r(m,u)$ enters A during or after stage u. Now suppose that $m \notin W_k^A$. For each $s \geqslant u$, no number less than $r(m,s)$ enters A during or after stage s, so if m were in $W_k^A[s]$ then it would be in W_k^A, and hence $m \notin W_k^A[s]$. Thus in this case $r(m,s) = 0$ for all $s \geqslant u$. So we see that $r(m,s) = r(m,u)$ for all $m \leqslant n$ and $s \geqslant u$, and hence $\gamma(n,s) = \gamma(n,u)$ for all $s \geqslant u$. □

Jockusch and Shore [193, 194] used the pseudo-jump machinery to establish a number of results. One application is a finite injury proof that there is a high incomplete c.e. Turing degree, a result first proved using the infinite injury method, as we will see in Section 2.14.3. The Jockusch-Shore proof of the existence of a high incomplete c.e. degree in fact produces a superhigh degree (where a set X is *superhigh* if $\emptyset'' \leqslant_{\mathrm{tt}} X'$), and runs as follows. Relativize the original Friedberg theorem that there is a noncomputable set W_e of low (or even superlow) Turing degree, to obtain an operator V_e such that for all Y we have that $Y <_{\mathrm{T}} V_e^Y$ and V_e^Y is (super)low over Y. Then use Theorem 2.12.3 to obtain a c.e. set A with $V_e^A \equiv_{\mathrm{T}} \emptyset'$. Then $\emptyset'$ is (super)low over A and $A <_{\mathrm{T}} \emptyset'$, and hence A is (super)high and incomplete. We will use this pseudo-jump technique again in Section 11.8. See Jockusch and Shore [193, 194], Downey and Shore [132], and Coles, Downey, Jockusch, and LaForte [73] for more on the general theory of pseudo-jump operators.

2.13 Coding and permitting

Coding and permitting are two basic methods for controlling the degrees of sets we build. Coding is a way of ensuring that a set A we build has degree at least that of a given set B. As the name implies, it consists of encoding the bits of B into A in a recoverable way. One simple way to to do this is to build A to equal $B \oplus C$ for some C, but in some cases we need to employ a more elaborate coding method. We will see an example in Proposition 2.13.1 below.

Permitting is a way of ensuring that a set A we build has degree at most that of a given set B. We will describe the basic form of c.e. permitting. For more elaborate forms of c.e. permitting, see Soare [366]. For a thorough exposition of the Δ_2^0 permitting method, see Miller [281].

Let B be a noncomputable c.e. set, and suppose we are building a c.e. set A to satisfy certain requirements while also ensuring that $A \leqslant_{\mathrm{T}} B$. At a stage s in the construction, we may want to put a number n into A. But instead of putting n into A immediately, we wait for a stage $t \geqslant s$ such that $B_{t+1} \upharpoonright n \neq B_t \upharpoonright n$. At that stage, we say that B *permits* us to put n into A. If we always wait for permissions of this kind before putting numbers into A, then we can compute A from B as follows. Given n, look for an s such that $B_s \upharpoonright n = B \upharpoonright n$. Then $n \in A$ iff $n \in A_s$. Of course, one has to worry whether a particular requirement can live with this permitting strategy. But suppose that we have a requirement $\mathcal{R}$ that provides us with infinitely many followers n, and will be satisfied as long as one such n enters A. Suppose that we never get a permission to put such an n into A. Then we can compute B as follows. Given m, wait for a stage s such that $\mathcal{R}$ provides us with a follower $n > m$. Since we are assuming we are never permitted to put n into A, we have $B \upharpoonright n = B_s \upharpoonright n$. In particular, $m \in B$ iff $m \in B_s$. Since we are assuming that B is noncomputable, some follower must eventually be permitted to enter A, and hence $\mathcal{R}$ is satisfied.

The following proof illustrates both coding and permitting. Recall that a coinfinite c.e. set A is simple if its complement does not contain any infinite c.e. set.

Proposition 2.13.1. *Every nonzero c.e. degree contains a simple set.*

Proof. Let B be a noncomputable c.e. set. We may assume that exactly one element x_s enters B at each stage s. We build a c.e. set $A \equiv_{\mathrm{T}} B$ to satisfy the simplicity requirements

$$\mathcal{R}_e : |W_e| = \infty \ \Rightarrow \ W_e \cap A \neq \emptyset.$$

At stage s, let $a_0^s < a_1^s < \cdots$ be the elements of $\overline{A_s}$. For each $e < s$, if $W_{e,s} \cap A_s = \emptyset$ and there is an $n \in W_{e,s}$ with $n > 3e$ and $n > x_s$, then put n into A. Also, put $a_{3x_s}^s$ into A.

The set A is clearly c.e. Each $\mathcal{R}_e$ puts at most one number into A, and this number must be greater than $3e$, while for each $x \in B$, the coding of B puts only one number into A, and this number is at least $3x$. Thus A is coinfinite.

For each n, if $B_s \upharpoonright n+1 = B \upharpoonright n+1$, then $n < x_t$ for all $t > s$, so n cannot enter A after stage s. As explained above, it follows that $A \leqslant_{\mathrm{T}} B$.

Given x, the set A can compute a stage s such that $a_i^s \in \overline{A}$ for all $i \leqslant 3x$. Then $a_{3x}^t = a_{3x}^s$ for all $t \geqslant s$, so $x \in B$ iff either $x \in B_s$ or $a_{3x}^s \in A$. Thus $B \leqslant_{\mathrm{T}} A$.

Finally, we need to show that each $\mathcal{R}_e$ is met. Suppose that W_e is infinite but $W_e \cap A = \emptyset$. Then we can computably find $3e < n_0 < n_1 < \cdots$ and $e \leqslant s_0 < s_1 < \cdots$ such that $n_i \in W_{e,s_i}$ for all i. None of the n_i are ever permitted by B, that is, $n_i \leqslant x_t$ for all i and $t \geqslant s_i$. So $B \upharpoonright n_i = B_{s_i} \upharpoonright n_i$ for all i, and hence B is computable, contrary to hypothesis. Thus each requirement is met, and hence A is simple. □

2.14 The infinite injury priority method

In this section, we introduce the infinite injury priority method. We begin by discussing the concept of priority trees, then give a few examples of constructions involving such trees. See Soare [366] for further examples.

2.14.1 Priority trees and guessing

The Friedberg-Muchnik construction used to prove Theorem 2.11.1 can be viewed another way. Instead of having a single strategy for each requirement, which is restarted every time a stronger priority requirement acts, we can attach multiple versions of each requirement to a *priority tree*, in this case the full binary tree $2^{<\omega}$.

Recall that the relevant requirements are the following.

$$\mathcal{R}_{2e} : \Phi_e^A \neq B.$$

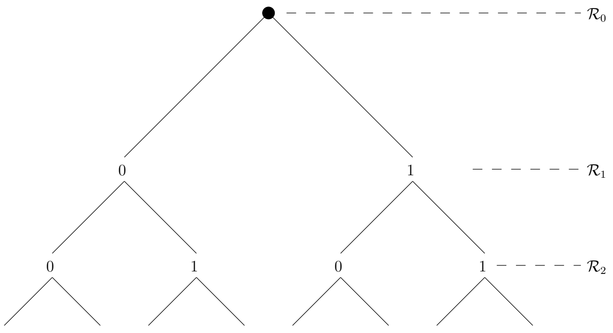
$$\mathcal{R}_{2e+1} : \Phi_e^B \neq A.$$

Attached to each node σ is a "version" R_σ of $\mathcal{R}_{|\sigma|}$. We call such a version a *strategy* for $\mathcal{R}_{|\sigma|}$, and refer to σ itself as a *guess*, for reasons that will soon become clear. Thus there are 2^e separate strategies for $\mathcal{R}_e$.

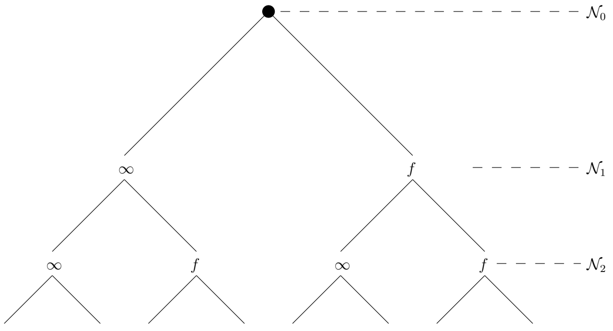
Each strategy R_σ has two *outcomes*, 0 and 1. The outcome 1 indicates that R_σ appoints a follower but never enumerates this follower into its target set. The outcome 0 indicates that R_σ actually enumerates its follower into its target set. At a stage s, we have a guess as to which outcome is correct, depending on whether or not R_σ has enumerated its follower into its target set. If this guess is ever 0, then we immediately know that 0 is indeed the correct outcome, so we regard the 0 outcome as being stronger than the 1 outcome. We order our guesses lexicographically, and hence if σ is to the left of τ, then we think of σ as being stronger than τ.

Figure 2.1 shows the above setup. For comparison, it also shows the setup for the Minimal Pair Theorem, which will be discussed in the next subsection.

Here is how this mechanism is used. Consider $\mathcal{R}_1$. We have two strategies for this requirement. One, which in this paragraph we write as $R_{\langle 0 \rangle}$ rather than R_0 to avoid confusion with $\mathcal{R}_0$, is below the 0 outcome of R_λ and the other, $R_{\langle 1 \rangle}$, is below the 1 outcome of R_λ. We think of these versions



For the Friedberg-Muchnik Theorem



For the Minimal Pair Theorem

Figure 2.1. The assignment of priorities and outcomes

as guessing that R_λ's outcome is 0 and 1, respectively. The strategy $R_{\langle 1\rangle}$ believes that R_λ will appoint a follower, but will never enumerate it into its target set. Thus its action is simply to act immediately after its guess appears correct, that is, once R_λ appoints its follower. At this point, $R_{\langle 1\rangle}$ can appoint its own follower, and proceed to act as $\mathcal{R}_1$ would have acted in the original proof of the Friedberg-Muchnik Theorem. Its belief may turn out to be false (i.e., R_λ may enumerate its follower into its target set), in which case its action may be negated, but in that case we do not care about $R_{\langle 1\rangle}$. Indeed, in general, we only care about strategies whose guesses about the outcomes of strategies above them in the tree are correct. So we have the "back-up" strategy $R_{\langle 0\rangle}$, which believes that R_λ will enumerate its follower into its target set. This strategy will act only once its guess appears correct, i.e., after R_λ enumerates its follower into its target set. Then and only then does this strategy wake up and appoint its follower.

The above is extended inductively on the tree of strategies in a natural way. For instance, for $\sigma = 0110$, there is a strategy R_σ for $\mathcal{R}_4$, which guesses that R_λ and R_{011} do not enumerate their followers, but R_0 and R_{01} do.

This strategy waits for R_λ and R_{011} to appoint followers, and for R_0 and R_{01} to both appoint followers and then enumerate them. Once all of these conditions are met, it appoints its own follower and proceeds to act as $\mathcal{R}_4$ would have acted in the original proof of the Friedberg-Muchnik Theorem.

More precisely, at stage s, we have a guess σ_s of length s, which is defined inductively. Only those strategies R_τ with $\tau \preccurlyeq \sigma_s$ get to act at stage s. We say that such τ are *visited* at stage s. Suppose we have defined $\sigma_s \upharpoonright n$. Let $\tau = \sigma_s \upharpoonright n$. Then we allow R_τ to act as follows. Let us suppose that $n = 2e$, the odd case being symmetric. If R_τ has previously put a number into B, then it does nothing. In this case, if $n < s$ then $\sigma(n) = 0$. Otherwise, R_τ acts in one of three ways.

1. If R_τ does not currently have a follower, then it appoints a follower x_τ greater than any number seen in the construction so far. In this case, if $n < s$ then $\sigma_s(n) = 1$.
2. Otherwise, if $\Phi_e^A(x_\tau)[s]\downarrow= 0 = B_s(x_\tau)$, then R_τ enumerates x_τ into B. In this case, if $n < s$ then $\sigma_s(n) = 0$.
3. Otherwise, R_τ does nothing. In this case, if $n < s$ then $\sigma_s(n) = 1$.

The strategy R_λ has a *true outcome* i, which is 0 if it eventually enumerates its witness into B, and 1 otherwise. Similarly, R_i has a true outcome. The *true path* TP of the construction is the unique path such that $\mathrm{TP}(n)$ is the true outcome of $R_{\mathrm{TP}\upharpoonright n}$. Note that, in the above construction, $\mathrm{TP}(n) = \lim_s \sigma_s(n)$, but with an eye to more complicated tree constructions, we in general define the true path of such a construction as the leftmost path visited infinitely often (that is, the path extending those guesses τ such that τ is the leftmost guess of length $|\tau|$ that is visited infinitely often). The idea behind this definition is that the strategies on the true path are the ones that actually succeed in meeting their respective requirements.

In the construction described above, it is easy to verify that this is indeed the case. Let $\tau \in \mathrm{TP}$. Again, let us assume that $|\tau| = 2e$, the odd case being symmetric. At the first stage s at which τ is visited, a follower x_τ is appointed. Since followers are always chosen to be fresh large numbers, x_τ will not be enumerated into B by any strategy other than R_τ. So if $\Phi_e^A(x_\tau)\uparrow$ or $\Phi_e^A(x_\tau)\downarrow= 1$, then $\mathcal{R}_{|\tau|}$ is met. Otherwise, there is a stage $t > s$ such that $\Phi_e^A(x_\tau)[t]\downarrow= 0 = B_t(x_\tau)$ and τ is visited at stage t. Then R_τ enumerates x_τ into B.

Nodes to the right of $\tau 0$ will never again be visited, so the corresponding strategies will never again act. Nodes to the left of τ are never visited at all, so the corresponding strategies never act. Nodes extending $\tau 0$ will not have been visited before stage t, so the corresponding strategies will pick followers greater than $\varphi_e^A(x_\tau)[t]$. If $n < |\tau|$ then there are two possibilities. If $\tau(n) = 1$ then 1 is the true outcome of $R_{\tau\upharpoonright n}$, and hence that strategy never enumerates its witness. If $\tau(n) = 0$ then $R_{\tau\upharpoonright n}$ must have already enumerated its witness by stage s, since otherwise τ would not have been visited at that stage. In either case, we see that strategies corresponding to nodes extended by τ do not enumerate numbers during or after stage t.

The upshot of the analysis in the previous paragraph is that no strategy enumerates a number less than $\varphi_e^A(x_\tau)[t]$ into A during or after stage t, so by the use principle, $\Phi_e^A(x_\tau)\downarrow = 0 \neq B(x_\tau)$, and hence $\mathcal{R}_\tau$ is met.

It is natural to wonder why we should introduce all this machinery. For the Friedberg-Muchnik Theorem itself, as well as for most finite injury arguments, the payoff in new insight provided by this reorganization of the construction does not really balance the additional notational and conceptual burden. However, the above concepts were developed in the past few decades to make *infinite* injury arguments comprehensible.

The key difference between finite injury and infinite injury arguments is the following. In an infinite injury argument, the action of a given requirements $\mathcal{R}$ may be infinitary. Obviously, we cannot simply restart weaker priority requirements every time $\mathcal{R}$ acts. Instead, we can have multiple strategies for weaker priority requirements, depending on whether or not $\mathcal{R}$ acts infinitely often. Thus, in a basic setup of this sort, the left outcome of $\mathcal{R}$, representing the guess that $\mathcal{R}$ acts infinitely often, is visited each time $\mathcal{R}$ acts.

In the Friedberg-Muchnik argument, the approximation to the true path moves only to the left as time goes on, since the guessed outcome of a strategy can change from 1 to 0, but never the other way. Thus, the true path is computable in $\emptyset'$. In infinite injury arguments, the approximation to the true path can move both left and right. Assuming that the tree is finitely branching, this possibility means that, in general, the true path is computable only in $\emptyset''$. It clearly is computable in $\emptyset''$ because, letting TP_s be the unique string of length s visited at stage s,

$$\sigma \prec \mathrm{TP} \text{ iff } \exists^\infty s\,(\sigma \prec \mathrm{TP}_s) \wedge \exists^{<\infty} s\,(\mathrm{TP}_s <_{\mathrm{lex}} \sigma).$$

In the following sections, we will give examples of infinite injury priority tree constructions to illustrate the application of this mechanism.

2.14.2 The minimal pair method

Our first example of infinite injury argument is the construction of a minimal pair of c.e. degrees. A pair of noncomputable degrees $\mathbf{a}$ and $\mathbf{b}$ is a *minimal pair* if the only degree below both $\mathbf{a}$ and $\mathbf{b}$ is $\mathbf{0}$.

Theorem 2.14.1 (Lachlan [231], Yates [408]). *There is a minimal pair of c.e. degrees.*

Proof. We build noncomputable c.e. sets A and B such that every set computable in both A and B is computable. We construct A and B in stages to satisfy the following requirements for each e.

$$\mathcal{R}_e : \overline{A} \neq W_e.$$

$$\mathcal{Q}_e : \overline{B} \neq W_e.$$

$$\mathcal{N}_e : \Phi_e^A = \Phi_e^B \text{ total } \Rightarrow \Phi_e^A \text{ computable.}^7$$

We arrange these requirements in a priority list as in previous constructions.

We meet the $\mathcal{R}$- and $\mathcal{Q}$-requirements by a Friedberg-Muchnik type strategy. For instance, to satisfy $\mathcal{R}_e$, we pick a follower x, targeted for A, and wait until x enters W_e. If this event never happens, then $x \in \overline{A} \cap \overline{W_e}$, so $\mathcal{R}_e$ is met. If x does enter W_e, then we put x into A, thus again meeting $\mathcal{R}_e$.

The $\mathcal{N}$-requirements are trickier to meet. We first discuss how to meet a single $\mathcal{N}_e$ in isolation, and then look at the coherence problems between the various requirements and the solution to these provided by the use of a tree of strategies.

We follow standard conventions, in particular assuming that all uses at stage s are bounded by s. As in the proof of the Sacks Splitting Theorem 2.12.1, we have a length of agreement function

$$l(e,s) = \max\{n : \forall k < n\,(\Phi_e^A(k)[s]\downarrow = \Phi_e^B(k)[s]\downarrow)\}$$

and an associated use function

$$u(e,s) = \max\{\varphi_e^A(l(e,t)-1)[t],\ \varphi_e^B(l(e,t)-1)[t] : t \leqslant s\},$$

using the convention that use functions are monotone increasing where defined. We now also have a maximum length of agreement function

$$m(e,s) = \max\{l(e,t) : t \leqslant s\},$$

which can be thought of as a high water mark for the length of agreements seen so far. We say that a stage s is *e-expansionary* if $l(e,s) > m(e,s-1)$, that is, if the current length of agreement surpasses the previous high water mark.

The key idea for meeting $\mathcal{N}_e$ is the following. Suppose that $n < l(e,s)$, so that $\Phi_e^A(n)[s]\downarrow = \Phi_e^B(n)[s]\downarrow$. Now suppose that we allow numbers to enter A at will, but "freeze" the computation $\Phi_e^B(n)[s]$ by not allowing

[7] Clearly, meeting the $\mathcal{R}$- and $\mathcal{Q}$-requirements is enough to ensure that A and B are noncomputable. As for the $\mathcal{N}$-requirements, it might seem at first glance that we need stronger requirements, of the form $\Phi_i^A = \Phi_j^B$ total $\Rightarrow$ Φ_i^A computable. However, suppose that we are able to meet the $\mathcal{N}$-requirements. Then we cannot have $A = B$, since the $\mathcal{N}$-requirements would then force A and B to be computable. So there is an n such that $A(n) \neq B(n)$. Let us assume without loss of generality that $n \in B$. If $f \leqslant_{\mathrm{T}} A, B$ then there are i and j such that $\Phi_i^A = \Phi_j^B = f$. But also, there is an e such that, for all oracles X, we have $\Phi_e^X = \Phi_i^X$ if $n \notin X$ and $\Phi_e^X = \Phi_j^X$ if $n \in X$. For this e, we have $\Phi_e^A = \Phi_e^B = f$. So if such an f is total, then $\mathcal{N}_e$ ensures that it is computable. This argument is known as *Posner's trick*. It is, of course, merely a notational convenience.

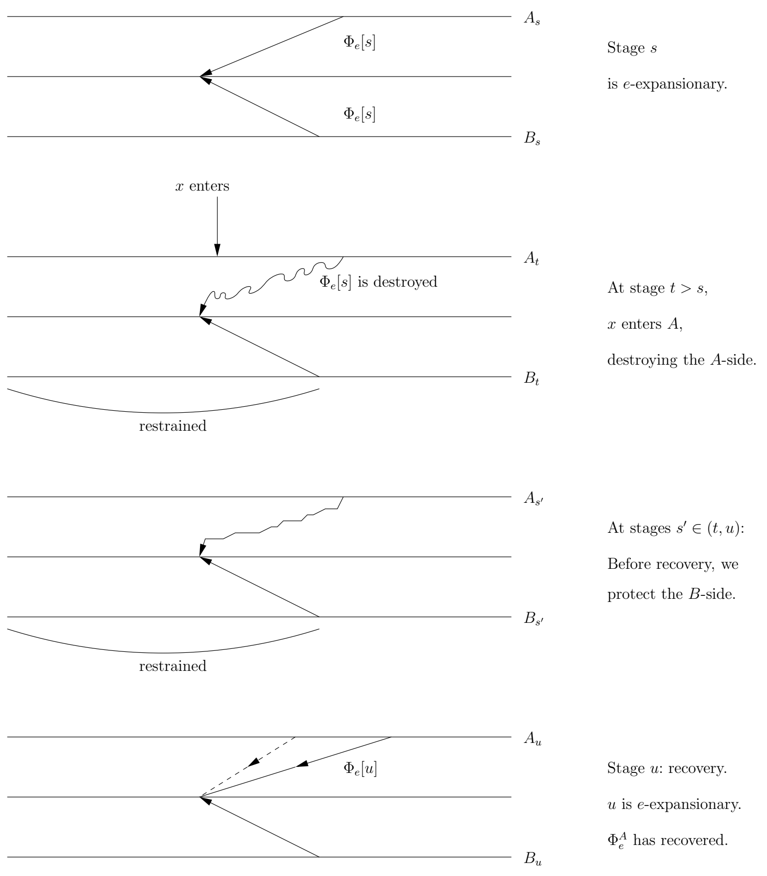


Figure 2.2. The $\mathcal{N}_e$ strategy

any number to enter B below $\varphi_e^B(n)[s]$ until the next e-expansionary stage $t > s$. At this stage t, we again have $\Phi_e^A(n)[t]\downarrow = \Phi_e^B(n)[t]\downarrow$. But we also have $\Phi_e^B(n)[t] = \Phi_e^B(n)[s]$, by the use principle. Now suppose that we start allowing numbers to enter B at will, but freeze the computation $\Phi_e^A(n)[t]$ by not allowing any number to enter A below $\varphi_e^A(n)[t]$ until the next e-expansionary stage $u > t$. Then again $\Phi_e^A(n)[u]\downarrow = \Phi_e^B(n)[u]\downarrow$, but also $\Phi_e^A(n)[u] = \Phi_e^A(n)[t] = \Phi_e^B(n)[t] = \Phi_e^B(n)[s] = \Phi_e^A(n)[s]$. (Note that we are not saying that these *computations* are the same, only that they have the same *value*. It may well be that $\varphi_e^A(n)[u] > \varphi_e^A(n)[s]$, for example.) So if we keep to this strategy, alternately freezing the A-side and the B-side of our agreeing computations, then we ensure that, if $\Phi_e^A = \Phi_e^B$ is total (which implies that there are infinitely many e-expansionary stages), then $\Phi_e^A(n) = \Phi_e^A(n)[s]$. Thus, if we follow this strategy for all n, then we can compute Φ_e^A, and hence $\mathcal{N}_e$ is met. Figure 2.2 illustrates this idea.

In summary, the strategy for meeting $\mathcal{N}_e$ is to wait until an e-expansionary stage s, then impose a restraint $u(e, s)$ on A, then wait until the next e-expansionary stage t, lift the restraint on A, and impose a restraint $u(e, t)$ on B, and then continue in this way, alternating which set is restrained at each new e-expansionary stage. Note that there is an important difference between this strategy and the one employed in the proof of the Sacks Splitting Theorem. There we argued that the length of agreement associated with a given requirement could not go to infinity. Here, however, it may well be the case that $\lim_s l(e, s) = \infty$, and hence $\lim_s u(e, s) = \infty$. Thus, the restraints imposed by the strategy for $\mathcal{N}_e$ may tend to infinity.

How do the $\mathcal{R}$- and $\mathcal{Q}$-requirements of weaker priority deal with this possibility? Consider a requirement $\mathcal{R}_i$ weaker than $\mathcal{N}_e$. We have two strategies for $\mathcal{R}_i$. One strategy guesses that there are only finitely many e-expansionary stages. This strategy picks a fresh large follower. Each time a new e-expansionary stage occurs, the strategy is *initialized*, which means that it must pick a new fresh large follower. Otherwise, the strategy acts exactly as described above. That is, it waits until its current follower x enters W_i, if ever, then puts x into A.

The other strategy for $\mathcal{R}_i$ guesses that there are infinitely many e-expansionary stages. It picks a follower x. If x enters W_i at stage s, then it wants to put x into A, but it may be restrained from doing so by the strategy for $\mathcal{N}_e$. However, if its guess is correct then there will be an e-expansionary stage $t \geqslant s$ at which the restraint on A is dropped. At that stage, x can be put into A.

Coherence. When we consider multiple $\mathcal{N}$-requirements, we run into a problem. Consider two requirements $\mathcal{N}_e$ and $\mathcal{N}_i$, with the first having stronger priority. Let $s_0 < s_1 < \cdots$ be the e-expansionary stages and $t_0 < t_1 < \cdots$ be the i-expansionary stages. During the interval $[s_0, s_1)$, weaker strategies are prevented from putting certain numbers into A by $\mathcal{N}_e$. At stage s_1, this restraint is lifted, but if $t_0 \leqslant s_1 < t_1$, then at stage s_1 it will be $\mathcal{N}_i$ that prevents weaker strategies from putting certain numbers into A. When $\mathcal{N}_i$ drops that restraint at stage t_1, we may have a new restraint on A imposed by $\mathcal{N}_e$ (if say $s_2 \leqslant t_1 < s_3$). Thus, although individually $\mathcal{N}_e$ and $\mathcal{N}_i$ each provide infinitely many stages at which they allow weaker strategies to put numbers into A, the two strategies *together* may conspire to block weaker strategies from *ever* putting numbers into A.

This problem is overcome by having two strategies for $\mathcal{N}_i$. The one guessing that there are only finitely many e-expansionary stages has no problems. The one guessing that there are infinitely many e-expansionary stages acts only at such stages, and in particular calculates its expansionary stages based only on such stages, which forces its expansionary stages to be nested within the e-expansionary stages. Thus the actions of $\mathcal{N}_e$ and $\mathcal{N}_i$ are forced to cohere.

We now turn to the formal details of the construction. These details may at first appear somewhat mysterious, in that it may be unclear how they actually implement the strategies described above. However, the verification section should clarify things.

The Priority Tree. We use the tree $\mathcal{T} = \{\infty, f\}^{<\omega}$. (Of course, this tree is the same as $2^{<\omega}$, but renaming the nodes should help in understanding the construction.) To each $\sigma \in \mathcal{T}$, we assign strategies N_σ, R_σ, and Q_σ for $\mathcal{N}_{|\sigma|}$, $\mathcal{R}_{|\sigma|}$, and $\mathcal{Q}_{|\sigma|}$, respectively. (It is only the $\mathcal{N}$-requirements that really need to be on the tree, since only they have outcomes we care about, but this assignment is notationally convenient.) Again we use lexicographic ordering, with ∞ to the left of f.

Definition 2.14.2. We define the notions of *σ-stage*, $m(\sigma, s)$, and *σ-expansionary stage* by induction on $|\sigma|$.

(i) Every stage is a λ-stage.

(ii) Suppose that s is a τ-stage. Let $e = |\tau|$. Let

$$m(\tau, s) = \max\{l(e,t) : t < s \text{ is a } \tau\text{-stage}\}.$$

If $l(e, s) > m(\tau, s)$ then we say that s is τ-expansionary and declare s to be a $\tau^\frown\infty$-stage. Otherwise, we declare s to be a $\tau^\frown f$-stage.

Let TP_s be the unique σ of length s such that s is a σ-stage.

Definition 2.14.3. We say that R_σ *requires attention* at a σ-stage $s > |\sigma|$ if $W_{|\sigma|,s} \cap A_s = \emptyset$ and one of the following holds.

(i) R_σ currently has no follower.

(ii) R_σ has a follower $x \in W_{|\sigma|,s}$.

The definition of Q_σ requiring attention is analogous.

Construction. At stage s, proceed as follows.
Step 1. Compute TP_s. Initialize all strategies attached to nodes to the right of TP_s. For the R- and Q-strategies, this initialization means that their followers are canceled.
Step 2. Find the strongest priority $\mathcal{R}$- or $\mathcal{Q}$-requirement that has a strategy requiring attention at stage s. Let us suppose that this requirement is $\mathcal{R}_e$ (the $\mathcal{Q}$-requirement case being analogous), and that R_σ is the corresponding strategy requiring attention at stage s. (Note that s must be a σ-stage.) We say that R_σ *acts* at stage s. Initialize all strategies attached to nodes properly extending σ. If R_σ does not currently have a follower, appoint a fresh large follower for R_σ. Otherwise, enumerate R_σ's follower into A.
End of Construction.

Verification. Let the true path TP be the leftmost path of $\mathcal{T}$ visited infinitely often. In other words, let TP be the unique path of $\mathcal{T}$ such that

$$\sigma \prec \mathrm{TP} \text{ iff } \exists^\infty s\,(\sigma \prec \mathrm{TP}_s) \wedge \exists^{<\infty} s\,(\mathrm{TP}_s <_{\mathrm{lex}} \sigma).$$

We write $\tau \leqslant \mathrm{TP}$ to mean that τ is either on or to the left of the true path.

Lemma 2.14.4. *Each $\mathcal{R}$- and $\mathcal{Q}$-requirement is met.*

Proof. Consider $\mathcal{R}_e$ (the $\mathcal{Q}$-requirement case being analogous). Let $\sigma = \mathrm{TP} \upharpoonright e$. Assume by induction that the strategies attached to proper prefixes of σ act only finitely often, and let s be the least stage such that

1. no strategy attached to a proper prefix of σ acts after stage s and
2. TP_t is not to the left of σ for all $t > s$.

By the minimality of s, it must be the case that R_σ is initialized at stage s. Thus, at the next σ-stage $t > s$, it will be assigned a follower x. Since R_σ cannot be initialized after stage t, this follower is permanent. By the way followers are chosen, x will not be put into A unless x enters W_e. If x enters W_e at stage $u > t$, then at the first σ-stage $v \geqslant u$, the strategy R_σ will act, and x will enter A. In any case, R_σ succeeds in meeting $\mathcal{R}_e$. Note that R_σ acts at most twice after stage s, and hence the induction can continue. □

Lemma 2.14.5. *Each $\mathcal{N}$-requirement is met.*

Proof. Consider $\mathcal{N}_e$. Let $\sigma = \mathrm{TP} \upharpoonright e$. If $\sigma^\frown f \prec TP$ then there are only finitely many e-expansionary stages, so $\Phi_e(A) \neq \Phi_e(B)$, and hence $\mathcal{N}_e$ is met.

So suppose that $\sigma^\frown\infty \prec TP$ and that Φ_e^A is total. We will show that Φ_e^A is computable. Let s be the least stage such that

1. no strategy attached to a prefix of σ (including σ itself) acts after stage s and
2. TP_t is not to the left of σ for all $t > s$.

To compute $\Phi_e^A(n)$, find the least $\sigma^\frown\infty$-stage $t_0 > s$ such that $l(e, s) > n$. We claim that $\Phi_e^A(n) = \Phi_e^A(n)[t_0]$.

Let $t_0 < t_1 < \cdots$ be the $\sigma^\frown\infty$-stages greater than or equal to t_0. Each such stage is e-expansionary, so we have $\Phi_e^A(n)[t_i] = \Phi_e^B(n)[t_i]$ for all i. We claim that, for each i, we have either $\Phi_e^A(n)[t_{i+1}] = \Phi_e^A(n)[t_i]$ or $\Phi_e^B(n)[t_{i+1}] = \Phi_e^B(n)[t_i]$. Assuming this claim, it follows easily that $\Phi_e^A(n)[t_0] = \Phi_e^A(n)[t_1] = \cdots$, which implies that $\Phi_e^A(n) = \Phi_e^A(n)[t_0]$.

To establish the claim, fix i. At stage t_i, we initialize all strategies to the right of $\sigma^\frown\infty$. Thus, any follower of such a strategy appointed after stage t_i must be larger than any number seen in the construction by stage t_i, and in particular larger than $\varphi_e^A(n)[t_i]$ and $\varphi_e^B(n)[t_i]$. By the choice of s, strategies above or to the left of σ do not act after stage s. Thus, the only strategies that can put numbers less than $\varphi_e^A(n)[t_i]$ into A or numbers less than $\varphi_e^B(n)[t_i]$ into B between stages t_i and t_{i+1} are the ones associated with extensions of $\sigma^\frown\infty$. But such a strategy cannot act except at a $\sigma^\frown\infty$-stage, and at each stage at most one strategy gets to act. Thus, at most one strategy associated with an extension of $\sigma^\frown\infty$ can act between stages t_i

and t_{i+1}, and hence only one of $A \upharpoonright \varphi_e^A(n)[t_i]$ and $B \upharpoonright \varphi_e^B(n)[t_i]$ can change between stages t_i and t_{i+1}. So either $A \upharpoonright \varphi_e^A(n)[t_{i+1}] = A \upharpoonright \varphi_e^A(n)[t_i]$ or $B \upharpoonright \varphi_e^B(n)[t_{i+1}] = B \upharpoonright \varphi_e^B(n)[t_i]$, which implies that either $\Phi_e^A(n)[t_{i+1}] = \Phi_e^A(n)[t_i]$ or $\Phi_e^B(n)[t_{i+1}] = \Phi_e^B(n)[t_i]$. As mentioned above, it follows that $\Phi_e^A(n) = \Phi_e^A(n)[t_0]$. $\square$

These two lemmas show that all requirements are met, which concludes the proof of the theorem. $\square$

Giving a full proof of the following result is a good exercise for those unfamiliar with the methodology above.

Theorem 2.14.6 (Ambos-Spies [5], Downey and Welch [135])**.** *There is a noncomputable c.e. set A such that if $C \sqcup D = A$ is a c.e. splitting of A, then the degrees of C and D form a minimal pair.*

Proof sketch. The proof is similar to that of Theorem 2.14.1, using the length of agreement function

$$\begin{aligned} l(i,j,k,m,s) = \max\{x : \forall y < x\, [\Phi_i^{W_k}(y)[s]\downarrow = \Phi_j^{W_m}(y)[s]\downarrow \\ \wedge\ \forall z \leqslant \varphi_i^{W_k}(y)[s], \varphi_j^{W_m}(y)[s]\, (W_k[s] \upharpoonright z \sqcup W_m[s] \upharpoonright z = A_s \upharpoonright z)]\}. \end{aligned}$$

$\square$

2.14.3 High computably enumerable degrees

Another example of an infinite injury argument is the construction of an incomplete high c.e. degree. Recall that a c.e. degree $\mathbf{a}$ is high if $\mathbf{a}' = \mathbf{0}''$. We begin with a few definitions. Recall that $A^{[e]}$ denotes the eth column $\{\langle e,n\rangle : \langle e,n\rangle \in A\}$ of A.

Definition 2.14.7. A set A is *piecewise trivial* if for all e, the set $A^{[e]}$ is either finite or equal to $\mathbb{N}^{[e]}$.

A subset B of A is *thick* if for all e we have $A^{[e]} =^* B^{[e]}$ (i.e., the symmetric difference of $A^{[e]}$ and $B^{[e]}$ is finite).

Lemma 2.14.8. (i) *There is a piecewise trivial c.e. set A such that $A^{[e]}$ is infinite iff Φ_e is total.*

(ii) *If B is a thick subset of such a set A, then B is high.*

Proof. (i) Let $A = \{\langle e,n\rangle : e \in \mathbb{N} \wedge \forall k \leqslant n\, \Phi_e(k)\downarrow\}$. Then A is c.e., and clearly $A^{[e]}$ is infinite iff $A^{[e]} = \mathbb{N}^{[e]}$ iff Φ_e is total.

(ii) Define a reduction Γ by letting $\Gamma^X(e,s) = 1$ if $\langle e,s\rangle \in X$ and $\Gamma^X(e,s) = 0$ otherwise. If Φ_e is total then $A^{[e]} = \mathbb{N}^{[e]}$, so $B^{[e]}$ is coinfinite, which implies that $\lim_s \Gamma^B(e,s) = 1$. On the other hand, if Φ_e is not total then $A^{[e]}$ is finite, so $B^{[e]}$ is finite, which implies that $\lim_s \Gamma^B(e,s) = 0$.

Thus, the function f defined by $f(e) = \lim_s \Gamma^B(e,s)$ is total. By the relativized form of the limit lemma, $f \leqslant_T B'$. So B' can decide whether a given Φ_e is total or not. By Theorem 2.6.5, $\emptyset'' \leqslant_T B'$. □

Thus, to construct an incomplete high c.e. degree, it suffices to prove the following result, which is a weak form of the Thickness Lemma discussed in the next section.

Theorem 2.14.9 (Shoenfield [355]). *Let C be a noncomputable c.e. set, and let A be a piecewise trivial c.e. set. Then there is a c.e. thick subset B of A such that $C \nleqslant_T B$.*

Proof. We construct $B \subseteq A$ to meet the following requirements for all e.

$$\mathcal{R}_e : |A^{[e]} \setminus B^{[e]}| < \infty.$$
$$\mathcal{N}_e : \Phi_e^B \neq C.$$

We give the intuition and formal details of the construction, and sketch out its verification, leaving some details to the reader.

To meet $\mathcal{R}_e$, we must make sure that almost all of the eth column of A gets into B. To meet $\mathcal{N}_e$, we use the strategy already employed in the proof of the Sacks Splitting Theorem 2.12.1. That is, we measure the length of agreement

$$l(e,s) = \max\{n : \forall k < n\,(\Phi_e^B(k)[s] = C_s(k))\},$$

with the idea of preserving B_s on the use $\varphi_e^B(n)[s]$ for all $n < l(e,s)$.

The problem comes from the interaction of this preservation strategy with the strategies for stronger priority $\mathcal{R}$-requirements, since these may be infinitary. It might be the case that we infinitely often try to preserve an agreeing computation $\Phi_e^B(n)[s] = C_s(n)$, only to have some $\langle i,x\rangle$ enter A with $i < e$ and $\langle i,x\rangle < \varphi_e^B(n)[s]$. Since we must put almost every such pair into B to meet $\mathcal{R}_i$, our strategy for meeting $\mathcal{N}_e$ might be injured infinitely often.

On the other hand, we do know that $\mathcal{R}_i$ can do only one of two things. Either $A^{[i]}$ is finite, and hence $\mathcal{R}_i$ stops injuring $\mathcal{N}_e$ after some time, or almost all of $A^{[i]}$ is put into B. In the latter case, the numbers put into B for the sake of $\mathcal{R}_i$ form a computable set (since A is piecewise trivial). It is easy to adapt the strategy for $\mathcal{N}_e$ to deal with this case. Let S be the computable set of numbers put into B for the sake of $\mathcal{R}_i$. We can then proceed with the Sacks preservation strategy, except that we do not believe a computation $\Phi_e^B(k)[s]$ unless B_s already contains every element of $S \upharpoonright \varphi_e^B(k)[s]$. This modification prevents the action taken for $\mathcal{R}_i$ from ever injuring the strategy for meeting $\mathcal{N}_e$.

Of course, we do not know which of the two possibilities for the action of $\mathcal{R}_i$ actually happens, so, as before, we will have multiple strategies for $\mathcal{N}_e$, representing guesses as to whether or not $A^{[i]}$ is finite. There are several ways to organize this argument as a priority tree construction.[8] We give the details of one such construction.

The N-strategies do not have interesting outcomes, so we use the same tree $\mathcal{T} = \{\infty, f\}^{<\omega}$ as in the proof of the Minimal Pair Theorem 2.14.1, with the same lexicographic ordering as before (i.e., the ordering induced by regarding ∞ as being to the left of f). To each $\sigma \in \mathcal{T}$ of length e we associate strategies R_σ and N_σ for meeting $\mathcal{R}_e$ and $\mathcal{N}_e$, respectively.

Definition 2.14.10. We define the notions of *σ-stage*, *σ-believable computation*, *σ-restraint $r(\sigma, s)$*, *σ-expansionary stage*, and *σ-length of agreement $l(\sigma, s)$* by induction on $|\sigma|$ and s as follows.

Every stage is a λ-stage, and $r(\sigma, 0) = 0$ for all σ. As usual, if we do not explicitly define $r(\sigma, s)$ at stage s, then $r(\sigma, s) = r(\sigma, s-1)$.

Suppose that $s > 0$ is a σ-stage, and let $e = |\sigma|$. A computation $\Phi_e^B(k)[s] = C_s(k)$ is σ-believable if for all $\tau^\frown\infty \preccurlyeq \sigma$ and all x, if $r(\tau, s) < \langle |\tau|, x\rangle < \varphi_e^B(k)[s]$, then $\langle |\tau|, x\rangle \in B_s$. Let

$$l(\sigma, s) = \max\{n : \forall k < n\, (\Phi_e^B(k)[s] = C_s(k) \text{ via a } \sigma\text{-believable computation})\}.$$

Say that s is σ-expansionary if $l(\sigma, s) > \max\{l(\sigma, t) : t < s\}$. Let

$$r(\sigma, s) = \max(\{r(\tau, s) : \tau <_{\text{lex}} \sigma\} \cup \{\varphi_e^B(k)[s] : k < l(\sigma, s)\}).$$

Let t be the last σ-stage before s, or 0 if there have been no such stages. If $|A_s^{[e]}| > |A_t^{[e]}|$ then s is a $\sigma^\frown\infty$-stage. Otherwise, s is a $\sigma^\frown f$-stage.

Construction. The construction is now rather simple. At stage s, let σ_s be the unique string of length s such that s is a σ_s-stage. For each $e < s$ and each $\langle e, x\rangle \in A_s$ that is not yet in B_s, if $\langle e, x\rangle > r(\sigma_s \upharpoonright i, s)$ then put $\langle e, x\rangle$ into B.
End of Construction.

We now sketch the verification that this construction succeeds in meeting all requirements. As usual, the true path TP of the construction is the leftmost path visited infinitely often.

Let $\sigma \in$ TP, let $e = |\sigma|$, and assume by induction that $\lim_t r(\tau, t)$ is well-defined for all $\tau \prec \sigma$. Let s be a stage such that

1. $r(\tau, t)$ has reached a limit $r(\tau)$ by stage s for every $\tau \prec \sigma$,
2. the construction never moves to the left of σ after stage s,
3. $B \upharpoonright r(\tau) = B_s \upharpoonright r(\tau)$ for every $\tau \prec \sigma$, and

[8] It is often the case in an infinite injury construction that we have substantive choices in defining the priority tree. In particular, some authors prefer to encode as much information as possible about the behavior of a strategy into its outcomes, while others prefer to encode only the information that weaker strategies need to have. It is worth keeping this fact in mind when reading such constructions.

4. $B^{[i]}_s = B^{[i]}$ for all $i < e$ such that $\sigma(i) = f$.

Note that item 4 makes sense because if $\sigma \in \mathrm{TP}$ and $\sigma(i) = f$, then there is a τ of length i such that $\tau^\frown f \in \mathrm{TP}$, which means that $A^{[i]}$ is finite, and hence so is $B^{[i]}$.

It is now not hard to argue, as in the proof of the Sacks Splitting Theorem, that if $t > s$ is a σ-stage and $n < l(\sigma, t)$, then the computation $\Phi^B_e(n)[s]$ is preserved forever. (The key fact is that this computation is σ-believable, and hence cannot be injured by stronger priority strategies.) Again as in the proof of the Sacks Splitting Theorem, it must be the case that $\lim_t l(\sigma, t)$ exists, and hence $\mathcal{N}_e$ is met.

It is also not hard to argue now that $r(\sigma) = \lim_t r(\sigma, t)$ is well-defined. Let u be a stage by which this limit has been reached. Then every $\langle e, x\rangle > r(\sigma)$ that enters A after stage u is eventually put into B, and hence $\mathcal{R}_e$ is met. □

A good exercise for those new to the techniques presented in this section is to combine the constructions of this subsection and the previous one to build a minimal pair of high c.e. degrees.

2.14.4 The Thickness Lemma

Theorem 2.14.9 is only a weak form of the real Thickness Lemma of Shoenfield [355]. To state the full version, we need a new definition.

Definition 2.14.11. A set A is *piecewise computable* if $A^{[e]}$ is computable for each e.

Theorem 2.14.12 (Thickness Lemma, Shoenfield [355])**.** *Let C be a noncomputable c.e. set, and let A be a piecewise computable c.e. set. Then there is a c.e. thick subset B of A such that $C \nleqslant_{\mathrm{T}} B$.*

Proof Sketch. We briefly sketch how to modify the proof of Theorem 2.14.9. Recall that in that result, we assumed that A was piecewise trivial. Now we have the weaker assumption that A is piecewise computable, that is, every column of A is computable. Thus, for each e, there is a c.e. set $W_{g(e)} \subseteq \mathbb{N}^{[e]}$ such that $W_{g(e)}$ is the complement of $A^{[e]}$ in $\mathbb{N}^{[e]}$ (meaning that $W_{g(e)} \sqcup A^{[e]} = \mathbb{N}^{[e]}$).

The key to how we used the piecewise triviality of A in Theorem 2.14.9 was in the definition of σ-believable computation. Recall that for σ with $e = |\sigma|$, we said that a computation $\Phi^B_e(k)[s] = C_s(k)$ at a σ-stage s was σ-believable if for all $\tau^\frown\infty \preccurlyeq \sigma$ and all x, if $r(\tau, s) < \langle |\tau|, x\rangle < \varphi^B_e(k)[s]$, then $\langle |\tau|, x\rangle \in B_s$. This definition relied on the fact that, if $\tau^\frown\infty \in \mathrm{TP}$, then every $\langle |\tau|, x\rangle > r(\tau, s)$ was eventually put into B. The corresponding fact here is that every $\langle |\tau|, x\rangle > r(\tau, s)$ that is not in $W_{g(|\tau|)}$ is eventually put into B.

So in order to adjust the definition of σ-believable computation, for each $\tau \prec \sigma$, we need to know an index j such that $W_j = W_{g(|\tau|)}$. Since we cannot compute such a j from $|\tau|$, we must guess it along the tree of strategies. That is, for each τ, instead of the outcomes ∞ and f, we now have an outcome j for each $j \in \mathbb{N}$, representing a guess that $W_j = W_{g(|\tau|)}$. The outcome j is taken to be correct at a stage s if j is the least number for which we see the length of agreement between $\mathbb{N}^{[e]}$ and $A^{[e]} \sqcup W_j$ increase at stage s.

Now a computation $\Phi_e^B(k)[s] = C_s(k)$ at a σ-stage s is σ-believable if for all $\tau^\frown j \preccurlyeq \sigma$ and all x, if $r(\tau, s) < \langle |\tau|, x\rangle < \varphi_e^B(k)[s]$, then $\langle |\tau|, x\rangle \in B_s \cup W_j$. The rest of the construction is essentially the same as before. One important point is that, because the tree of strategies is now infinitely branching, the existence of the true path is no longer automatically guaranteed. However, if τ is visited infinitely often then $\tau^\frown j$ is visited infinitely often for the least j such that $W_j = W_{g(|\tau|)}$, from which it follows that TP is well-defined. There are infinite injury constructions in which showing that TP is well-defined requires a careful proof. □

We make some remarks for those who, like the senior author, were brought up with the "old" techniques using the "hat trick" and the "window lemma" of Soare [366]. It is rather ironic that the very first result that used the infinite injury method in its proof was the Thickness Lemma, since, like the Density Theorem of the next section, it has a proof *not* using priority trees that is combinatorially much easier to present. In a sense, this fact shows an inherent shortcoming of the tree technique in that often more information needs to be represented on the tree than is absolutely necessary for the proof of the theorem. To demonstrate this point, and since it is somewhat instructive, we now sketch the original proof of the Thickness Lemma.

We have the same requirements, but we define $\widehat{\Phi}^B(n)[s]$ to be $\Phi^B(n)[s]$ unless some number less than $\varphi^B(n)[s]$ enters B at stage s, in which case we declare that $\widehat{\Phi}^B(n)[s]\uparrow$. This is called the *hat convention*. Using this convention, we can generate the hatted length of agreement $\widehat{l}(e, s)$ with $\widehat{\Phi}$ in place of Φ, and so forth. Finally, we define the *restraint function*

$$\widehat{r}(e, s) = \max\{\widehat{\varphi}^B(n)[s] : n < \widehat{l}(e, s)\}.$$

The construction is to put $\langle e, x\rangle \in A_s$ into B at stage s if it is not yet there and $\langle e, x\rangle > \max\{\widehat{r}(j, s) : j \leqslant e\}$.

To verify that this construction succeeds in meeting all our requirements, it is enough to show that, for each e, the *injury set*

$$\widehat{I}_{e,s} = \{x : \exists v \leqslant s\, (x \leqslant \widehat{r}(e, v) \wedge x \in B_{s+1} \setminus B_v)\}$$

is computable, $B^{[e]} =^* A^{[e]}$, and $\mathcal{N}_e$ is met, all by simultaneous induction, the key idea being the "window lemma", which states that the liminf of the restraint $\widehat{R}(e, s) = \max\{\widehat{r}(j, s) : j \leqslant e\}$ is finite. The key point is that to verify these facts we do not actually need to know during the construction what the complement of $A^{[e]}$ is. This knowledge is used only in the verification. See Soare [366, Theorem VIII.1.1] for more details.

There is a strong form of the Thickness Lemma that is implicit in the work of Lachlan, Robinson, Shoenfield, Sacks, Soare, and others. We write $A^{[<e]}$ for $\bigcup_{i<e} A^{[i]}$.

Theorem 2.14.13 (Strong Thickness Lemma, see Soare [366]). *Let C be such that $\emptyset <_{\mathrm{T}} C \leqslant \emptyset'$ and let A be c.e. There is a c.e. set $B \subseteq A$ such that $B \leqslant_{\mathrm{T}} A$ and for all e, if $C \nleqslant_{\mathrm{T}} A^{[<e]}$ then for all $j \leqslant e$ we have $C \neq \Phi_j^B$ and $B^{[j]} =^* A^{[j]}$. It follows that if $C \nleqslant_{\mathrm{T}} A^{[<e]}$ for all e then $C \nleqslant_{\mathrm{T}} B$ and B is a thick subset of A. Furthermore, an index for B can be obtained uniformly from indices for A and C.*

Theorem 2.14.13 can be proved with much the same methods as those used to prove the Thickness Lemma. Soare [366] showed that many results of classical computability theory can be obtained from the Strong Thickness Lemma. Here is an example.

Corollary 2.14.14 (Sacks [342]). *Let $\mathbf{a}_0 < \mathbf{a}_1 < \cdots$ be an infinite ascending sequence of uniformly c.e. degrees. Then there is an incomplete c.e. degree $\mathbf{b}$ such that $\mathbf{a}_0 < \mathbf{a}_1 < \cdots < \mathbf{b}$. Thus $\mathbf{0}'$ is not a minimal upper bound for any such sequence of degrees.*

Proof. Let h be a computable function such that $\mathbf{a}_i = \deg(W_{h(i)})$ for all i. Let $A = \{\langle i, n\rangle : n \in W_{h(i)}\}$. Let $C = \emptyset'$ and apply the Strong Thickness Lemma to get a c.e. set $B \subseteq A$ such that $C \nleqslant_{\mathrm{T}} B$ and $B^{[i]} =^* A^{[i]}$ (which implies that $\deg(B^{[i]}) = \mathbf{a}_i$) for all i. Let $\mathbf{b} = \deg(B)$. $\square$

2.15 The Density Theorem

One of the classic applications of the infinite injury method is to extend the Friedberg-Muchnik Theorem to show that not only are there intermediate c.e. degrees, but in fact the c.e. degrees form a dense partial ordering. It is possible to give a short and elegant proof of this result without using priority trees (see [366]), but it is quite hard to understand how that proof works. Therefore, in this section we give a proof using trees.

Theorem 2.15.1 (Sacks Density Theorem [344]). *For any c.e. sets $B <_{\mathrm{T}} C$, there are c.e. sets $A_0 \mid_{\mathrm{T}} A_1$ such that $B <_{\mathrm{T}} A_i <_{\mathrm{T}} C$ for $i = 0, 1$.*

Proof. We will construct A_0, A_1 satisfying the following requirements for all e.

$$\mathcal{R}_{2e} : \Phi_e^{A_0} \neq A_1.$$
$$\mathcal{R}_{2e+1} : \Phi_e^{A_1} \neq A_0.$$

Additionally, we must meet the global permitting requirement

$$A_0, A_1 \leqslant_{\mathrm{T}} C$$

and the global coding requirement

$$B \leqslant_{\mathrm{T}} A_0, A_1.$$

Together with the incomparability of the A_i, these global requirements imply that $B <_{\mathrm{T}} A_i <_{\mathrm{T}} C$ for $i = 0, 1$.

To satisfy the coding requirement, we simply code B into the even bits of each A_i. That is, whenever n enters B, we put $2n$ into A_i, and we ensure that all other numbers entering A_i are odd.

To satisfy the $\mathcal{R}$-requirements, we adopt a Friedberg-Muchnik type strategy, which we later modify to cope with the coding and permitting requirements. Consider the satisfaction of $\mathcal{R}_{2e}$. We pick an odd follower n, wait until $\Phi_e^{A_0}(n)[s]\downarrow= 0$, and if that ever happens, put n into A_1 and attempt to preserve $A_0 \upharpoonright \varphi_e^{A_0}(n)[s]$. This strategy is not correct as stated, but will serve as a basis for future modifications. The basic problems are that we cannot control B and that we can put numbers into A_1 only when permitted to do so by C. We deal with the latter issue first.

In place of a single follower n, we will use a potentially infinite collection $\{n(i) : i \in \mathbb{N}\}$ of followers or *coding markers*, picked inductively as follows. Suppose that we have already chosen $n(j)$ for $j < i$, and that all of these have been realized (a concept that will be defined below). We choose a large fresh odd follower $n(i)$ targeted for A_1. Now we wait until $\Phi_e^{A_0}(n(i))[s]\downarrow= 0$. If that ever happens, we declare $n(i)$ to be *realized*, attempt to preserve $\varphi_e^{A_0}(n(i))[s]$ by initializing weaker priority requirements, and repeat this procedure for $i + 1$.

We keep to this procedure until i enters C for some realized $n(i)$, at which point we put $n(i)$ into A_1. Without considering the effect of B or stronger priority strategies, this strategy will be successful, because if infinitely many followers are realized, then eventually C must permit us to put one of them into A_1, since otherwise we could compute C.

Now let us consider the effect of the coding of B into A_0. After we realize a follower $n(i)$ at stage s, we could be forced to change $A_0 \upharpoonright \varphi_e^{A_0}(n(i))[s]$ by a change in B. We could attempt to cope with this possibility by simply declaring $n(i)$ to be unrealized and waiting for a new stage t such that $\Phi_e^{A_0}(n(i))[t]\downarrow= 0$. We could then declare $n(i)$ to be once again realized and attempt to preserve $\varphi_e^{A_0}(n(i))[t]$ by initializing weaker priority requirements. The problem is that again we could be forced to change $A_0 \upharpoonright \varphi_e^{A_0}(n(i))[t]$ by a change in B. (Notice that $\varphi_e^{A_0}(n(i))[t]$ could be much larger than $\varphi_e^{A_0}(n(i))[s]$.) If we keep to this strategy, we could infinitely often realize $n(i)$, then unrealize it, then realize it again, and so on. Of course, in this case, $\mathcal{R}_{2e}$ is satisfied because $\Phi_e^{A_0}(n(i))\uparrow$. Unfortunately, weaker strategies are in trouble, as they are initialized infinitely often. In other words, we have won the $\mathcal{R}_{2e}$-battle, but lost the war.

The solution to this dilemma is to notice that this ugly situation can happen only if the values of $\varphi_e^{A_0}(n(i))[s]$ are unbounded for some i. If we *knew* this to be the case, then we could do the construction ignoring $\mathcal{R}_{2e}$, as it would automatically be satisfied. Obviously, though, we cannot know whether this situation happens ahead of time. However, we can have an infinitary outcome for $\mathcal{R}_{2e}$, which *guesses* that this use is unbounded. The trick is to make the effect on the construction of this outcome computable, hence allowing strategies below it to live with the action of $\mathcal{R}_{2e}$ if it truly has this infinitary outcome.

To make this idea more precise, we begin by presenting the basic module for the $\mathcal{R}_{2e}$ strategy described above, in the absence of stronger priority requirements. (Thus the single strategy for $\mathcal{R}_0$, the strongest requirement, acts exactly as follows.) The basic module for $\mathcal{R}_{2e+1}$ strategies is of course symmetric.

1. If all currently defined followers are realized, then choose a large fresh odd follower $n(i)$.

2. If $\Phi_e^{A_0}(n(i))[s]\downarrow = 0$, then declare $n(i)$ to be realized and initialize weaker strategies.

3. If $n(i)$ is realized and i enters C, then put $n(i)$ into A_1. In this case, stop choosing new followers unless $n(i)$ later becomes unrealized.

4. If $n(i)$ is realized and the coding of B causes A_0 to change below $\varphi_e^{A_0}(n(i))[t]$ at stage t, then, for the least i for which this is the case at this stage, cancel all $n(j)$ for $j > i$ (which will have different values if defined again later) and declare $n(i)$ to be unrealized.

Note that at each stage s, the strategy for $\mathcal{R}_{2e}$ has realized followers $n(0), \dots, n(i-1)$ and an unrealized follower $n(i)$. For each realized follower $n(j)$, we have $\Phi_e^{A_0}(n(j))[s]\downarrow$.

To see that the above module succeeds in satisfying a single requirement $\mathcal{R}_{2e}$, suppose not, so that $\Phi_e^{A_0} = A_1$. Then each $n(i)$ is eventually defined and reaches a final value, since all uses of the form $\varphi_e^{A_0}(n(i))$ eventually settle. Moreover, we can compute these final values using B, since it is only the coding of B into A_0 that can change these uses. Now we claim that B can compute C, contrary to hypothesis. Given i, we can use B to compute a stage s at which $n(i)$ reaches its final value and becomes permanently realized. If i were to enter C after stage s, then $n(i)$ would enter A_1, ensuring that $\Phi_e^{A_0} \neq A_1$. Thus $i \in C$ iff $i \in C_s$.

The above also shows that this module has two possible kinds of outcomes. Either $\varphi_e^{A_0}(n(i))$ is unbounded for some i, or $\Phi_e^{A_0}(n(i)) \neq A_1(n(i))$. We denote the former outcome by (i, u) and the latter by (i, f). We use these outcomes to generate the priority tree $\mathcal{T} = \{(i,u), (i,f) : i \in \mathbb{N}\}^{<\omega}$, with a lexicographic ordering inherited from the ordering of outcomes $(i,u) <_L (i,f) <_L (i+1,u)$.

We now discuss how weaker priority requirements can live in the environments provided by $\mathcal{R}_{2e}$, by considering the case $e = 0$. The markers $n(i)$ for the $\mathcal{R}_0$ strategy will now be denoted by $n(\lambda, i)$ (since that strategy is associated with the empty sequence in $\mathcal{T}$). There are infinitely many strategies for $\mathcal{R}_1$, one for each outcome. We denote the strategy below outcome σ by R_σ and its markers by $n(\sigma, i)$.

A strategy living below (i, f) guesses that the strategy for $\mathcal{R}_0$ has finite action. Hence it can be initialized each time we realize $n(\lambda, j)$ for $j \leqslant i$, and simply implements the basic module.

A strategy R_σ living below $\sigma = (i, u)$ guesses that $n(\lambda, i)$ comes to a final value but no $n(\lambda, j)$ for $j > i$ does, because we keep changing A_0 below $\varphi_0^{A_0}(n(\lambda, i))[s]$ for the sake of coding B. As in the Thickness Lemma, this strategy will believe a $\Phi_0^{A_1}$-computation only if the current value of $n(\lambda, i+1)$ (and hence every $n(\lambda, j)$ for $j > i$) is greater than the use of that computation.

However, there is a problem here. The strategy R_σ may want to put numbers into A_0 smaller than uses that the $\mathcal{R}_0$ strategy is trying to preserve. For instance, say a follower $n(\sigma, j)$ is appointed, then followers $n(\lambda, k)$ with $k > i$ are appointed and become realized with corresponding uses larger than $n(\sigma, j)$, then $n(\sigma, j)$ becomes realized and permitted by C. If the guess that (i, u) is the true outcome of the $\mathcal{R}_0$ strategy is correct, then there is no problem. We can put $n(\sigma, j)$ into A_0 and still satisfy $\mathcal{R}_0$. But otherwise, we cannot afford to allow the enumeration of $n(\sigma, j)$ to injure $\mathcal{R}_0$.

We now come to the main idea of this proof, Sacks' notion of *delayed permitting*: Since R_σ is guessing that (i, u) is the true outcome of the $\mathcal{R}_0$ strategy, it believes that any use larger than $n(\sigma, j)$ that the $\mathcal{R}_0$ strategy is trying to preserve will eventually be violated by the coding of B into A_0. So once $n(\sigma, j)$ is realized and C-permitted, instead of immediately putting $n(\sigma, j)$ into A_0, we merely promise to put it there if these uses do go away; that is, if R_σ becomes accessible again before $n(\sigma, j)$ is initialized.

This delayed permitting still allows C to compute A_0, because C can compute B, and hence know whether the relevant uses will ever be violated. That is, given a realized follower $n(\sigma, j)$, the set C knows whether $j \in C$. If not, then $n(\sigma, j)$ never enters A_0. Otherwise, C can check what the B-conditions are for R_σ to be accessible again, find out whether these conditions will ever obtain, and if so, go to that stage of the construction to check whether $n(\sigma, j)$ has not been initialized in the meantime, which is the only case in which $n(\sigma, j)$ enters A_0.

It is a subtle fact that, while the true path of the construction remains computable only from $\mathbf{0}''$, the fate of a particular follower is always computable from C. Furthermore, we can still argue that, if $\sigma = (i, u)$ is in fact the true outcome of $\mathcal{R}_0$ strategy and R_σ fails, then B can compute C, in much the same way as before, since every C-permitted follower eventually has no restraint placed on it.

Construction.

We code B by enumerating $2n$ into A_0 and A_1 whenever n enters B.

At a stage s, the construction proceeds in substages $t \leqslant s$. At each substage t, some node $\alpha(s,t)$ of $\mathcal{T}$ will be accessible, beginning with $\alpha(s,0) = \lambda$. At the end of the stage (which might come before substage s), we will have defined a node α_s. At substage t, the strategy $R_{\alpha(s,t)}$ for $\mathcal{R}_t$ will act. Let us assume that $t = 2e$, the odd case being symmetric, and let $\alpha = \alpha(s,t)$. We say that s is an *α-stage.*

If $n(\alpha,0)$ is currently undefined, then proceed as follows. Define $n(\alpha,0)$ to be a fresh large odd number. Let $\alpha_s = \alpha(s,t+1) = \alpha(s,t)^\frown(0,f)$, initialize all $\tau \not\leqslant_{\mathrm{L}} \alpha_s$, and end the stage.

We say that the computation $\Phi_e^{A_0}(x)[s]$ is *α-correct* if for all $\beta^\frown(j,u) \preccurlyeq \alpha$, if $n(\beta,j)$ is currently defined, then it is greater than $\varphi_e^{A_0}[s]$. Let $\ell(\alpha,s)$ be the α-correct length of agreement at stage s, that is, the largest m such that for all $x < m$, we have $\Phi_e^{A_0}(x)[s] = A_1(x)[s]$ via an α-correct computation. Adopt the first among the following cases that obtains.

1. For some i with $n(\alpha,i)$ realized (defined below), i has entered C since the last α-stage, and the α-correct length of agreement has exceeded $n(\alpha,i)$ since the last α-stage. Fix the least such i. Let $\alpha_s = \alpha(s,t+1) = \alpha(s,t)^\frown(i,f)$, and initialize all $\tau \not\leqslant_{\mathrm{L}} \alpha_s$. Put $n(\alpha,i)$ into A_0 and end the stage.

2. For some i with $n(\alpha,i)$ realized, the computation $\Phi_e^{A_0}(n(\alpha,i))$ has been injured since the previous α-stage. Fix the least such i. Declare $n(\alpha,i)$ to be unrealized. Cancel all $n(\alpha,j)$ for $j > i$. Let $\alpha(s,t+1) = \alpha^\frown(i,u)$. Initialize all τ with $\tau \not\leqslant_{\mathrm{L}} \alpha(s,t+1)$ and $\alpha(s,t+1) \not\preccurlyeq \tau$. If $t < s$ then proceed to the next substage. Otherwise let $\alpha_s = \alpha(s,t+1)$ and end the stage.

3. For some unrealized $n(\alpha,i)$, we have $\ell(\alpha,s) > n(\alpha,i)$. Fix the least such i. Declare $n(\alpha,i)$ to be realized, define $n(\alpha,i+1)$ to be a fresh large odd number, and cancel all $n(\alpha,j)$ for $j > i+1$. Let $\alpha_s = \alpha(s,t+1) = \alpha^\frown(i+1,f)$. Initialize all $\tau \not\leqslant_{\mathrm{L}} \alpha_s$ and end the stage.

4. Otherwise. Find the least (necessarily unrealized) $n(\alpha,i)$ that is defined. Let $\alpha(s,t+1) = \alpha^\frown(i,f)$. Initialize all τ with $\tau \not\leqslant_{\mathrm{L}} \alpha(s,t+1)$ and $\alpha(s,t+1) \not\preccurlyeq \tau$. If $t < s$ then proceed to the next substage. Otherwise let $\alpha_s = \alpha(s,t+1)$ and end the stage.

End of Construction.

Verification. We first show that the true path of the construction is well-defined, and all requirements are met. We say that α is on the *true path* if there are infinitely many α-stages but only finitely many s with $\alpha_s <_{\mathrm{L}} \alpha$.

If there are infinitely many α stages, then the *true outcome* of R_α is the leftmost outcome γ such that there are infinitely many α-stages s with $\alpha(s,|\alpha|+1) = \alpha^\frown\gamma$.

Lemma 2.15.2. *Let α be on the true path, and let γ be its true outcome. Then R_α succeeds in meeting $\mathcal{R}_{|\alpha|}$, there are infinitely many α-stages where the stage does not end at substage $|\alpha|$, and R_α initializes strategies below $\alpha^\frown\gamma$ only finitely often.*

Proof. Assume by induction that the lemma holds for strategies above α. Then, since α can be initialized only when $\alpha_s <_{\mathrm{L}} \alpha$ or when a strategy above it initializes it, there is a stage s_0 such that α is not initialized after stage s_0. After this stage, no $n(\alpha, i)$ is canceled except for the sake of an $n(\alpha, j)$ with $j < i$. Now we can argue as in the basic module, but using α-correct computations at α-stages, to show that R_α succeeds in meeting $\mathcal{R}_{|\alpha|}$.

If $\gamma = (i, u)$ then R_α does not initialize strategies below $\alpha^\frown\gamma$, and at every stage $s > |\alpha|$ at which $\alpha(s, |\alpha| + 1) = \alpha^\frown\gamma$, the stage does not end at substage $|\alpha|$.

If $\gamma = (i, f)$, then the R_α-module acts only finitely often. □

Thus, by induction, for each e there is an α of length e such that there are infinitely many α-stages but only finitely many s with $\alpha_s <_{\mathrm{L}} \alpha$, whence R_e is met.

Lemma 2.15.3. $A_0, A_1 \leqslant_{\mathrm{T}} C$.

Proof. We do the A_0 case, the A_i case being symmetric.

If $n \in A_0$, then n must be chosen by stage n as $n(\alpha, i)$ for some α and i with $|\alpha|$ even and $i \in C$. If that is the case, let s be the stage at which i enters C. Then n will be in A_0 iff there is an α-stage $t \geqslant s$ at which $n(\alpha, i)$ is still equal to n. So we may assume that s is not itself an α-stage and $n(\alpha, i)$ is not canceled at stage s, since otherwise we would already know whether $n \in A_0$ at stage s. Thus $\alpha <_{\mathrm{L}} \alpha_s$. Let β be the longest common initial segment of α and α_s. We have two cases.

First suppose that $\beta^\frown(k, f) \preccurlyeq \alpha$ for some k. Then $\beta^\frown(j, o) \preccurlyeq \alpha_s$ for some $j > k$ and $o \in \{f, u\}$. Suppose there is a least $\beta^\frown(k, f)$-stage $t > s$. If the construction moves to the left of $\beta^\frown(k, f)$ between stages s and t then α is initialized, so $n \notin A_0$. Otherwise, either case 1 or case 3 above holds for β at stage t, so again α is initialized and $n \notin A_0$. Of course, if there is no $\beta^\frown(k, f)$-stage after s, then $n \notin A_o$. Thus, in this case, we know that $n \notin A_0$ no matter what happens after stage s.

Now suppose that $\beta^\frown(k, u) \preccurlyeq \alpha$ for some k. After stage s, there can be an α-stage only if there is a $\beta^\frown(k, u)$-stage. Now C can use B to figure out whether the use of the relevant computation $\Phi_e^{A_i}(n(\beta, k))$ is B-correct. If it is B-correct, then $n \notin A_0$, since the only way for there to be a further $\beta^\frown(k, u)$-stage is for a strategy above or to the left of $\beta^\frown(k, u)$ to enumerate something into A_i, which causes α to be initialized. If not, then go to a stage t where B changes below this use. If t is not a $\beta^\frown(k, u)$-stage and $n(\alpha, i)$ is not canceled at stage t, then $\alpha_t <_{\mathrm{L}} \beta$. Let β' be the longest common initial segment of β and α_t. We can now repeat the above argument with β' in place of β.

It is a straightforward exercise to show that, proceeding in this manner, we eventually reach a stage at which we either can be sure that $n \notin A_0$, or we have an α-stage, at which point n enters A_0. Since we do so using B and $B \leqslant_{\mathrm{T}} C$, we have $A_0 \leqslant_{\mathrm{T}} C$. □

The two lemmas above complete the proof of the theorem. □

2.16 Jump theorems

We will see that the jump operator plays an important role in the study of algorithmic randomness. In this section, we look at several classic results about the range of the jump operator, whose proofs are combinations of techniques we have already seen. Of course, if $\mathbf{d} = \mathbf{a}'$ then $\mathbf{d} \geqslant \mathbf{0}'$. The following result is a converse to this fact.

Theorem 2.16.1 (Friedberg Completeness Criterion [160]). *If* $\mathbf{d} \geqslant \mathbf{0}'$ *then there is a degree* $\mathbf{a}$ *such that* $\mathbf{a}' = \mathbf{a} \vee \mathbf{0}' = \mathbf{d}$.

Proof. This is a finite extension argument. Let $D \in \mathbf{d}$. We construct a set A in stages, and let $\mathbf{a} = \deg(A)$. At stage 0, let $A_0 = \lambda$.

At odd stages $s = 2e+1$, we force the jump (i.e., decide whether $e \in A'$). We ask $\emptyset'$ whether there is a $\sigma \succ A_{s-1}$ such that $\Phi_e^{\sigma}(e)\downarrow$. If so, we search for such a σ and let $A_s = \sigma$. If not, we let $A_s = A_{s-1}$. Note that, in this case, we have ensured that $\Phi_e^{A}(e)\uparrow$.

At even stages $s = 2e+2$, we code $D(e)$ into A by letting $A_s = A_{s-1}D(e)$.

This concludes the construction. We have $A \oplus \emptyset' \leqslant_{\mathrm{T}} A'$ (which is always the case for any set A), so we can complete the proof by showing that $A' \leqslant_{\mathrm{T}} D$ and $D \leqslant_{\mathrm{T}} A \oplus \emptyset'$.

Since $\emptyset' \leqslant_{\mathrm{T}} D$, we can carry out the construction computably in D. To decide whether $e \in A'$, we simply run the construction until stage $2e+1$. At this stage, we decide whether $e \in A'$. Thus $A' \leqslant_{\mathrm{T}} D$.

A $\emptyset'$ oracle can tell us how to obtain A_{2e+1} given A_{2e}, while an A oracle can tell us how to obtain A_{2e+2} given A_{2e+1}, so using $A \oplus \emptyset'$, we can compute A_{2n+2} for any given n. Since $D(n)$ is the last element of A_{2n+2}, we can compute $D(n)$ using $A \oplus \emptyset'$. Thus $D \leqslant_{\mathrm{T}} A \oplus \emptyset'$. □

The above proof should be viewed as a combination of a coding argument and the construction of a low set, done simultaneously. This kind of combination is a recurrent theme in the proofs of results such as the ones in this section.

The following extension of the above result is due to Posner and Robinson [315]. We give a proof due to Jockusch and Shore [194].

Theorem 2.16.2 (Posner and Robinson [315])**.** *r If* $\mathbf{c} > \mathbf{0}$ *and* $\mathbf{d} \geqslant \mathbf{0}' \vee \mathbf{c}$ *then there is a degree* $\mathbf{a}$ *such that* $\mathbf{a}' = \mathbf{a} \vee \mathbf{c} = \mathbf{d}$.

Proof. By Proposition 2.13.1, there is an immune $C \in \mathbf{c}$. (Recall that the complement of a simple set is immune.) Let $D \in \mathbf{d}$. We build a set A by finite extensions.

Let $A_0 = \lambda$. Given A_n, proceed as follows. Let $S_n = \{x : \exists \sigma \succcurlyeq A_n 0^x 1 \, (\Phi_n^\sigma(n) \downarrow)\}$. If S_n is finite then there is an $x \in C \setminus S_n$. In this case, let $A_{n+1} = A_n 0^x 1 D(n)$. If S_n is infinite then, since S_n is c.e. and C is immune, there is an $x \in S_n \setminus C$. In this case, let s be least such that $\exists \sigma \succcurlyeq A_n 0^x 1 \, (\Phi_n^\sigma(n)[s] \downarrow)$, let σ be the length-lexicographically least string witnessing this existential, and let $A_{n+1} = \sigma D(n)$.

It is easy to see that from the sequence $A_0, A_1, \ldots$ we can compute the set A', the set $A \oplus C$, and the set D. Thus, it is enough to show that this sequence can be computed from each of these three sets. Suppose we have computed A_n.

Since $D \geqslant_T \emptyset' \oplus C$, using D we can compute S_n and C and hence determine A_{n+1}.

If we have A', then we have A, so we can determine the x such that $A_n 0^x 1 \prec A$. Then using $\emptyset'$ we can determine whether $x \in S_n$, and hence determine A_{n+1}.

If we have $A \oplus C$, then again we can determine the x such that $A_n 0^x 1 \prec A$. By construction, $x \in S_n$ iff $x \notin C$, so we can then use C to determine A_{n+1}. □

It is not hard to show that if $A \leqslant_T \emptyset'$ then A' is c.e. in $\emptyset'$. Since also $\emptyset' \leqslant_T A'$, we say that A' is *computably enumerable in and above* $\emptyset'$, abbreviated by CEA($\emptyset'$). The following theorem is an elaboration of Theorem 2.16.1.

Theorem 2.16.3 (Shoenfield Jump Inversion Theorem [354])**.** *If D is CEA($\emptyset'$) then there is an $A \leqslant_T \emptyset'$ such that $A' \equiv_T D$.*

We will briefly sketch a proof of the following stronger result. (We restrict ourselves to a sketch because, from a modern viewpoint, the proof contains no new ideas beyond what we have seen so far. See Soare [366] for a full proof.)

Theorem 2.16.4 (Sacks Jump Inversion Theorem [341])**.** *If D is CEA($\emptyset'$) then there is a c.e. set A such that $A' \equiv_T D$.*

Proof sketch. Let D be CEA($\emptyset'$). Then D is Σ_2^0, so there is an approximation $\{D_s\}_{s \in \omega}$ such that $n \in D$ iff there is an s such that $n \in D_t$ for all $t > s$. Arguing as in the proof of Lemma 2.14.8, we have a c.e. set B such that $B^{[e]}$ is an initial segment of $\mathbb{N}^{[e]}$ and equals $\mathbb{N}^{[e]}$ iff $e \notin D$.

We need to make the jump of A high enough to compute D, which we do by meeting for each e the requirement

$$\mathcal{P}_e : A^{[e]} =^* B^{[e]}.$$

As in Lemma 2.14.8, this action ensures that A' can compute D. We also need to keep the jump of A computable in D, which we do by controlling the jump as in Theorem 2.11.2.

For $\mathcal{P}_e$, we enumerate all elements of $B^{[e]}$ into A as long as we are not restrained from doing so. This requirement has nodes on the priority tree $\mathcal{T}$ with outcomes $\{\infty, f\}$. The first outcome corresponds to the case $B^{[e]} = \mathbb{N}^{[e]}$, and the second to the case $B^{[e]} =^* \emptyset$. Naturally, in the end we need to argue that if the outcome on the true path is $\nu^\frown\infty$, then almost all of $B^{[e]}$ is enumerated into A. We refer to $\mathcal{P}_e$ nodes as *coding nodes.*

A strategy N corresponding to a lowness requirement has a finite number of coding nodes above it with which it needs to contend. The strategy N does not have to worry about coding nodes with outcome f above it. But suppose there is a coding node P corresponding to $\mathcal{P}_e$ outcome ∞ above N. Then N assumes that all of $\mathbb{N}^{[e]}$ enters A, except for numbers below the maximum r of the restraints imposed by strategies above P. Thus N does not believe any computation until it has seen all numbers in $\mathbb{N}^{[e]}$ between r and the use of the computation enter $A^{[e]}$.

The full construction works in the standard inductive way. As with the density theorem, while D cannot figure out the true path of the construction, it can sort out the fate of a given coding marker, by deciding whether a particular restraint is B-correct. □

The above result can be extended to higher jumps as follows.

Theorem 2.16.5 (Sacks [341]). *If D is CEA($\emptyset^{(n)}$) then there is a c.e. set A such that $A^{(n)} \equiv_{\mathrm{T}} D$.*

Proof. This result is proved by induction. Suppose that D is CEA($\emptyset^{(n+1)}$). Then we know that there is a set B that is CEA($\emptyset^{(n)}$) with $B' \equiv_{\mathrm{T}} D$, by the relativized form of Theorem 2.16.4. By induction, there is a c.e. set A with $A^{(n)} \equiv_{\mathrm{T}} B$. Then $A^{(n+1)} \equiv_{\mathrm{T}} D$. □

There are even extensions of the above result to transfinite ordinals in place of n.

We will later have occasion to use the fact that the proof of the Sacks Jump Inversion Theorem is uniform. That is, an index for the set A in Theorem 2.16.5 can be found effectively from an index for the c.e. operator enumerating D from $\emptyset^{(n)}$.

The Sacks Jump Inversion and Density Theorems have been extraordinarily influential. They have been extended and generalized in many ways. Roughly speaking, any "reasonable" set of requirements on the ordering of a collection of c.e. degrees and their jumps that does not explicitly contradict obvious partial ordering considerations (such as $\mathbf{a} \leqslant \mathbf{b} \Rightarrow \mathbf{a}' \leqslant \mathbf{b}'$) is realizable. Here is one example.

Theorem 2.16.6 (Jump Interpolation Theorem, Robinson [333]). *Let C and D be c.e. sets with $C <_{\mathrm{T}} D$, and let S be CEA(D) with $C' \leqslant_{\mathrm{T}} S$. Then there is a c.e. set A such that $C <_{\mathrm{T}} A <_{\mathrm{T}} D$ and $A' \equiv_{\mathrm{T}} S$.*

The methods for proving the Jump Interpolation Theorem are much the same as the ones we have seen. One can even prove vast generalizations for n-jumps with additional ordering requirements. For the latest word on this subject, see Lempp and Lerman [238, 239].

2.17 Hyperimmune-free degrees

The notions of hyperimmune and hyperimmune-free degree have many applications in the study of computability theory and its interaction with algorithmic randomness. There are several ways to define these notions. We will adopt what is probably the simplest definition, using the concept of domination. Recall that a function f dominates a function g if $f(n) \geqslant g(n)$ for almost all n. It is sometimes technically useful to work with the following closely related concept. A function f *majorizes* a function g if $f(n) \geqslant g(n)$ for *all* n.

Definition 2.17.1 (Miller and Martin [282]). A degree $\mathbf{a}$ is *hyperimmune* if there is a function $f \leqslant_{\mathrm{T}} \mathbf{a}$ that is not dominated by any computable function (or, equivalently, not majorized by any computable function). Otherwise, $\mathbf{a}$ is *hyperimmune-free*.

While $\mathbf{0}$ is clearly hyperimmune-free, all other Δ^0_2 degrees are hyperimmune, as shown by the following result.

Proposition 2.17.2 (Miller and Martin [282]). *If $\mathbf{a} < \mathbf{b} \leqslant \mathbf{a}'$ for some $\mathbf{a}$, then $\mathbf{b}$ is hyperimmune. In particular, every nonzero degree below $\mathbf{0}'$, and hence every nonzero c.e. degree, is hyperimmune.*

Proof. We do the proof for the case $\mathbf{a} = \mathbf{0}$. The general result follows by a straightforward relativization. Let B be a set such that $\emptyset <_{\mathrm{T}} B \leqslant_{\mathrm{T}} \emptyset'$. Let $g(n) = \mu s \geqslant n\, (B_s \upharpoonright n = B \upharpoonright n)$. Notice that $g(n)$ is *not* the stage s by which the approximation to $B \upharpoonright n$ has stabilized (so that $B_t \upharpoonright n$ is correct for all $t \geqslant s$), but rather the first stage s at which $B_s \upharpoonright n$ is correct. Clearly, $g \leqslant_{\mathrm{T}} B$.

We claim that no computable function majorizes g. Suppose that h is computable and majorizes g. Then we claim that B is computable. To compute $B \upharpoonright m$, search for an $n > m$ such that $B_t \upharpoonright m = B_n \upharpoonright m$ for all $t \in [n, h(n)]$. Such an n must exist because there is a stage at which the approximation to $B \upharpoonright m$ stabilizes. By the definition of g and the choice of h, we have $g(n) \in [n, h(n)]$, so $B \upharpoonright m = B_{g(n)} \upharpoonright m = B_n \upharpoonright m$. Thus B is computable, which is a contradiction. □

On the other hand, there do exist nonzero hyperimmune-free degrees.

Theorem 2.17.3 (Miller and Martin [282]). *There is a nonzero hyperimmune-free degree.*

Proof. We define a noncomputable set A of hyperimmune-free degree, using a technique known as *forcing with computable perfect trees*. A *function tree* is a function $T : 2^{<\omega} \to 2^{<\omega}$ such that $T(\sigma 0)$ and $T(\sigma 1)$ are incompatible extensions of $T(\sigma)$. For a function tree T, let $[T]$ be the collection of all X for which there is a $B \in 2^\omega$ such that $T(\sigma) \prec X$ for all $\sigma \prec B$. We build a sequence $\{T_i\}_{i\in\omega}$ of computable function trees such that $[T_0] \supseteq [T_1] \supseteq [T_2] \supseteq \cdots$. Since each $[T_i]$ is closed, $\bigcap_n [T_n] \neq \emptyset$. We will take A to be any element of this intersection. At stage $2e+1$, we ensure that $A \neq W_e$, while at stage $2e+2$, we ensure that Φ_e^A is majorized by a computable function.

At stage 0, we let T_0 be the identity map.

At stage $2e+1$, we are given T_{2e}. Let $i \in \{0,1\}$ be such that $T_{2e}(i) \neq W_e \upharpoonright |T_{2e}(i)|$. Since $T_{2e}(0)$ and $T_{2e}(1)$ are incompatible, such an i must exist. Now define T_{2e+1} by letting $T_{2e+1}(\sigma) = T_{2e}(i\sigma)$. Notice that $[T_{2e+1}]$ consists exactly of those elements of $[T_{2e}]$ that extend $T_{2e}(i)$, and hence, if $A \in [T_{2e+1}]$ then $A \neq W_e$.

At stage $2e+2$, we are given a computable function tree T_{2e+1}. It suffices to build T_{2e+2} to ensure that either

(i) if $A \in [T_{2e+2}]$ then Φ_e^A is not total, or

(ii) there is a computable function f such that if $A \in [T_{2e+2}]$ then $\Phi_e^A(n) \leqslant f(n)$ for all n.

First suppose there are n and σ such that $\Phi_e^{T_{2e+1}(\tau)}(n)\uparrow$ for all $\tau \succcurlyeq \sigma$. Then define T_{2e+2} by letting $T_{2e+2}(\nu) = T_{2e+1}(\sigma\nu)$. This definition clearly ensures that if $A \in [T_{2e+2}]$ then $\Phi_e^A(n)\uparrow$.

Now suppose there are no such n and σ. Then define T_{2e+2} as follows. For the empty string λ, search for a σ_λ such that $\Phi_e^{T_{2e+1}(\sigma_\lambda)}(0)\downarrow$ and define $T_{2e+2}(\lambda) = T_{2e+1}(\sigma_\lambda)$. Having defined σ_τ, search for incompatible extensions $\sigma_{\tau 0}$ and $\sigma_{\tau 1}$ of σ_τ such that $\Phi_e^{T_{2e+1}(\sigma_{\tau i})}(|\tau|)\downarrow$ for $i = 0, 1$, and define $T_{2e+2}(\tau i) = T_{2e+1}(\sigma_{\tau i})$. This process ensures that for all n and all τ of length n, we have $\Phi_e^{T_{2e+2}(\tau)}(n)\downarrow$. Furthermore, it ensures that T_{2e+2} is computable, so that we can define a computable function f by letting $f(n) = \max\{\Phi_e^{T_{2e+2}(\tau)}(n) : |\tau| = n\}$. Thus if $A \in [T_{2e+2}]$ then $\Phi_e^A(n) \leqslant f(n)$. □

Note that the above construction can be carried out effectively using $\emptyset''$ as an oracle, so there are nonzero Δ_3^0 hyperimmune-free degrees. It is also possible to adapt the construction to prove the following result.

Theorem 2.17.4 (Miller and Martin [282]). *There are continuum many hyperimmune-free degrees.*

On the other hand, we will see in Section 8.21.1 that the class of sets of hyperimmune-free degree has measure 0.

The name "hyperimmune degree" comes from the notion of a hyperimmune set, which we now define.[9] A *strong array* is a computable collection of pairwise disjoint finite sets $\{F_i\}_{i \in \mathbb{N}}$ (which means not only that the F_i are uniformly computable, but also that the function $i \mapsto \max F_i$ is computable). An infinite set A is *hyperimmune* if for all strong arrays $\{F_i\}_{i \in \mathbb{N}}$, there is an i such that $F_i \subset \overline{A}$. A c.e. set is *hypersimple* if its complement is hyperimmune.

Given the terminology, one would expect that a hyperimmune degree is one that contains a hyperimmune set. We will show that this is indeed the case by first introducing another equivalent characterization of the hyperimmune degrees. The *principal function* of a set $A = \{a_0 < a_1 < \cdots\}$ is the function p_A defined by $p_A(n) = a_n$.

Lemma 2.17.5 (Miller and Martin [282]). *A degree* $\mathbf{a}$ *is hyperimmune iff* $\mathbf{a}$ *contains a set* A *such that* p_A *is not majorized by any computable function.*

Proof. Since $p_A \leqslant_T A$, the "if" direction is obvious. For the other direction, suppose that there is a function $f \leqslant_T \mathbf{a}$ that is not majorized by any computable function. We may assume that f is increasing. Let $B \in \mathbf{a}$ and let $b_0 < b_1 < \cdots$ be the elements of B. Let $A = \{f(b_n) : n \in \mathbb{N}\}$. Using the fact that f is increasing, it is easy to show that $A \equiv_T B$. Thus $A \in \mathbf{a}$, and $p_A(n) \geqslant f(n)$ for all n, so p_A is not majorized by any computable function. □

Theorem 2.17.6 (Kuznecov [230], Medvedev [264], Uspensky [393]). *A degree is hyperimmune iff it contains a hyperimmune set.*

Proof. Suppose that A is not hyperimmune. Then there is a strong array $\{F_i\}_{i \in \mathbb{N}}$ such that $A \cap F_i \neq \emptyset$ for all i. Let $f(n) = \max \bigcup_{i \leqslant n} F_i$. Then f is computable, and $p_A(n) \leqslant f(n)$ for all n. So if no set in $\mathbf{a}$ is hyperimmune, then it follows from Lemma 2.17.5 that $\mathbf{a}$ is not hyperimmune.

Now suppose that $\mathbf{a}$ is not hyperimmune and let $A \in \mathbf{a}$. By Lemma 2.17.5, there is a computable function f such that $p_A(n) \leqslant f(n)$ for all n. Let $F_0 = [0, f(0)]$. Given F_i, let $k_i = \max F_i + 1$ and let $F_{i+1} = [k_i, f(k_i)]$. Then $p_A(k_i) \in A \cap F_i$ for all i, so A is not hyperimmune. □

The following is a useful characterization of the hyperimmune-free degrees.

[9]Some authors have argued that the adoption of the term "hyperimmune-free degree" is a historical accident, and that a more descriptive one, based on domination properties, should be used. Nies, Barmpalias, and Soare, for example, have all used the term *computably dominated.* Earlier, authors such as Gasarch and Simpson used the term *almost recursive.*

Proposition 2.17.7 (Jockusch [187], Martin [unpublished]). *The following are equivalent for a set A.*

(i) *A has hyperimmune-free degree.*

(ii) *For all functions f, if $f \leqslant_{\mathrm{T}} A$ then $f \leqslant_{\mathrm{tt}} A$.*

(iii) *For all sets B, if $B \leqslant_{\mathrm{T}} A$ then $B \leqslant_{\mathrm{tt}} A$.*

Proof. To show that (i) implies (ii), suppose that A has hyperimmune-free degree and $f = \Phi_e^A$, and let $g(n)$ be the number of steps taken by the computation $\Phi_e^A(n)$. Then $g \leqslant_{\mathrm{T}} A$, so there is a computable function h that majorizes g. Let Ψ be the functional defined as follows. On oracle X and input n, run the computation $\Phi_e^X(n)$ for $h(n)$ many steps. If this computation converges, then output its value, and otherwise output 0. Then Ψ is a truth table functional, and $\Psi^A = \Phi_e^A = f$.

Clearly (ii) implies (iii). To show that (iii) implies (i), assume that A does not have hyperimmune-free degree and let $g \leqslant_{\mathrm{T}} A$ be a function not dominated by any computable function. We build a set $B \leqslant_{\mathrm{T}} A$ such that $B \nleqslant_{\mathrm{tt}} A$. Let $\sigma_0, \sigma_1, \dots$ be an effective list of all truth tables. For each e and n, proceed as follows. If $\Phi_e(\langle e,n\rangle)[g(n)]\downarrow$ then let $\langle e,n\rangle \in B$ iff $A \nvDash \sigma_{\Phi_e(\langle e,n\rangle)}$. Note that this definition ensures that Φ_e does not witness a truth table reduction from A to B. If $\Phi_e(\langle e,n\rangle)[g(n)]\uparrow$ then let $\langle e,n\rangle \in B$. Clearly $B \leqslant_{\mathrm{T}} A$. Assume for a contradiction that $B \leqslant_{\mathrm{tt}} A$. Then there is an e such that Φ_e witnesses this fact. This Φ_e is total, so there is a computable function h such that $\Phi_e(\langle e,n\rangle)[h(n)]\downarrow$ for all n. Since h does not dominate g, there is an n such that $\Phi_e(\langle e,n\rangle)[g(n)]\downarrow$. For this n, we have $\langle e,n\rangle \in B$ iff $A \nvDash \sigma_{\Phi_e(\langle e,n\rangle)}$, contradicting the choice of e. □

The above result cannot be extended to wtt-reducibility. Downey [99] constructed a noncomputable c.e. set A such that for all sets B, if $B \leqslant_{\mathrm{T}} A$ then $B \leqslant_{\mathrm{wtt}} A$. Since A is noncomputable and c.e., it has hyperimmune degree.

2.18 Minimal degrees

The proof of Theorem 2.17.3 uses ideas that go back to a fundamental theorem of Spector [374].

Definition 2.18.1. A degree $\mathbf{a} > \mathbf{0}$ is *minimal* if there is no degree $\mathbf{b}$ with $\mathbf{0} < \mathbf{b} < \mathbf{a}$.

Theorem 2.18.2 (Spector [374]). *There is a minimal degree below $\mathbf{0}''$.*

Proof. As with Theorem 2.17.3, the proof uses forcing with computable perfect trees. As before, we build a sequence $\{T_i\}_{i\in\omega}$ of computable function trees such that $[T_0] \supseteq [T_1] \supseteq [T_2] \supseteq \cdots$, and take our set A of minimal degree to be any element of $\bigcap_n [T_n]$.

At stage 0, we let T_0 be the identity map.

The action at stage $2e+1$ is exactly the same as in the proof of Theorem 2.17.3. That is, we let $i \in \{0,1\}$ be such that $T_{2e}(i) \neq W_e \upharpoonright |T_{2e}(i)|$ and define T_{2e+1} by letting $T_{2e+1}(\sigma) = T_{2e}(i\sigma)$. As before, this action ensures that $A \neq W_e$.

At stage $2e+2$, we are given T_{2e+1} and want to meet the requirement

$$\mathcal{R}_e : \ \Phi_e^A \text{ total } \Rightarrow \ \Phi_e^A \equiv_{\mathrm{T}} \emptyset \ \vee \ A \leqslant_{\mathrm{T}} \Phi_e^A.$$

The first thing we do is follow the method used in the proof of Theorem 2.17.3 to try to force the nontotality of Φ_e^A. We ask $\emptyset''$ whether there are n and σ such that $\Phi_e^{T_{2e+1}(\tau)}(n)\uparrow$ for all $\tau \succcurlyeq \sigma$. If so, we define T_{2e+2} by letting $T_{2e+2}(\nu) = T_{2e+1}(\sigma\nu)$. This definition clearly ensures that $\Phi_e^A(n)\uparrow$, and hence that $\mathcal{R}_e$ is met.

If there are no such n and σ, then we try to ensure that if Φ_e^A is total then it is computable. The critical question to ask $\emptyset''$ is whether there is a σ such that for all n and all $\tau_0, \tau_1 \succcurlyeq \sigma$, if $\Phi_e^{T_{2e+1}(\tau_i)}(n)\downarrow$ for $i = 0,1$, then $\Phi_e^{T_{2e+1}(\tau_0)}(n) = \Phi_e^{T_{2e+1}(\tau_1)}(n)$. If this question has a positive answer, then we define T_{2e+2} by letting $T_{2e+2}(\nu) = T_{2e+1}(\sigma\nu)$. Then, to compute $\Phi_e^A(n)$ (assuming $\Phi_e^A(n)\downarrow$), we simply need to find a μ such that $\Phi_e^{T_{2e+2}(\mu)}(n)\downarrow$, and we are guaranteed that $\Phi_e^A(n) = \Phi_e^{T_{2e+2}(\mu)}(n)$. Thus, in this case, Φ_e^A is computable if total, and hence $\mathcal{R}_e$ is met.

If the answer to the above question is negative, then we need to ensure that $A \leqslant_{\mathrm{T}} \Phi_e^A$. The crucial new idea is Spector's notion of an *e-splitting tree.*[10]

Definition 2.18.3. We say that a function tree T is *e-splitting* if for each σ, there is an n such that $\Phi_e^{T(\sigma 0)}(n)\downarrow \neq \Phi_e^{T(\sigma 1)}(n)\downarrow$.

The crucial fact about e-splitting trees is the following.

Lemma 2.18.4. *Let T be e-splitting and let $B \in [T]$. If Φ_e^B is total then $\Phi_e^B \geqslant_{\mathrm{T}} B$.*

Proof. Suppose we are given Φ_e^B, and that Φ_e^B is total. Search for an n_0 such that $\Phi_e^{T(0)}(n_0)\downarrow \neq \Phi_e^{T(1)}(n_0)\downarrow$. Let i be such that $\Phi_e^B(n_0) = \Phi_e^{T(i)}(n_0)$. Then we know that $T(i) \prec B$. Now find an n_1 such that $\Phi_e^{T(i0)}(n_1)\downarrow \neq \Phi_e^{T(i1)}(n_1)\downarrow$. Let j be such that $\Phi_e^B(n_1) = \Phi_e^{T(ij)}(n_1)$. Then $T(ij) \prec B$. Continuing in this way, we can compute more and more of B. Thus $B \leqslant_{\mathrm{T}} \Phi_e^B$. □

So to meet $\mathcal{R}_e$, it is enough to ensure that T_{2e+2} is e-splitting. But recall that we are now in the case in which for each σ there are n and

[10] We are being a bit free and easy with history. The notion of being e-splitting was first used by Spector, but the influential use of *trees* in the construction of a minimal degree actually first occurred in Shoenfield's book [356].

$\tau_0, \tau_1 \succ \sigma$ such that $\Phi_e^{T_{2e+1}(\tau_0)}(n)\downarrow \neq \Phi_e^{T_{2e+1}(\tau_1)}(n)\downarrow$. So we can define T_{2e+2} as follows. Let $T_{2e+2}(\lambda) = T_{2e+1}(\lambda)$. Suppose we have defined $T_{2e+2}(\mu)$ to equal $T_{2e+1}(\sigma)$ for some σ. Search for τ_0 and τ_1 as above and define $T_{2e+2}(\mu i) = T_{2e+1}(\tau_i)$ for $i = 0, 1$. Then T_{2e+2} is computable and e-splitting. □

As with hyperimmune-free degrees, there are in fact continuum many minimal degrees. A crucial difference between hyperimmune-free degrees and minimal degrees is given by the following result. (We say that a degree **b** *bounds* a degree **a** if $\mathbf{a} \leqslant \mathbf{b}$.)

Theorem 2.18.5 (Yates [410], Cooper [76]). *Every nonzero c.e. degree bounds a minimal degree.*

In particular, there are minimal degrees below $\mathbf{0}'$ (which was first shown by Sacks [342]), and hence hyperimmune minimal degrees. The method used to prove Theorem 2.18.5, known as the *full approximation method*, is quite involved and would take us a little too far afield. See Lerman [240] or Odifreddi [311] for more details.

Minimal degrees form a very interesting class. We finish by quoting two important results about their possible Turing degrees.

Theorem 2.18.6 (Jockusch and Posner [192]). *All minimal degrees are* GL_2. *That is, if* $\mathbf{a}$ *is a minimal degree then* $\mathbf{a}'' \leqslant (\mathbf{a} \vee \mathbf{0}')'$.

This result improved an earlier one by Cooper [77], who showed that no minimal degree below $\mathbf{0}'$ can be high. (There are low minimal degrees, as well as minimal degrees below $\mathbf{0}'$ that are not low; see [192] for a discussion of these and related results.) Cooper [77] also proved the following definitive result.

Theorem 2.18.7 (Cooper [77]). *If* $\mathbf{b} \geqslant \mathbf{0}'$ *then there is a minimal degree* $\mathbf{a}$ *with* $\mathbf{a}' = \mathbf{b}$.

Thus there is an analog of the Friedberg Completeness Criterion (Theorem 2.16.1) for minimal degrees. It is not possible to prove an analog of the Shoenfield Jump Inversion Theorem since Downey, Lempp, and Shore [124] showed that there are degrees CEA($\emptyset'$) and low over $\mathbf{0}'$ that are not jumps of minimal degrees below $\mathbf{0}'$. Cooper [78] claimed a characterization of the degrees CEA($\emptyset'$) that are jumps of minimal degrees below $\mathbf{0}'$, but no details were given.

2.19 Π_1^0 and Σ_1^0 classes

2.19.1 Basics

A *tree* is a subset of $2^{<\omega}$ closed under initial segments. A *path* through a tree T is an infinite sequence $P \in 2^\omega$ such that if $\sigma \prec P$ then $\sigma \in T$. The collection of paths through T is denoted by $[T]$. A subset of 2^ω is a Π^0_1 *class* if it is equal to $[T]$ for some computable tree T.[11]

An equivalent formulation is that C is a Π^0_1 class if there is a computable relation R such that

$$C = \{A \in 2^\omega : \forall n\, R(A \restriction n)\}.$$

It is not hard to show that this definition is equivalent to the one via computable trees.

There is a host of natural examples of important Π^0_1 in several branches of logic. For example, let A and B be disjoint c.e. sets. The collection of *separating sets* $\{X : X \supseteq A \wedge X \cap B = \emptyset\}$ is a Π^0_1 class. An important special case is an *effectively inseparable* pair, that is, a pair of disjoint c.e. sets A and B for which there is a computable function f such that for all disjoint c.e. $W_e \supseteq A$ and $W_j \supseteq B$, we have $f(e,j) \notin W_e \cup W_j$. This is the two variable version of a creative set. One example of an effectively inseparable pair is the sets $\{e : \Phi_e(e)\downarrow= 0\}$ and $\{e : \Phi_e(e)\downarrow= 1\}$. The function f witnessing the effective inseparability of this pair is not hard to build: given e and j, using the recursion theorem we can find an i such that $\Phi_i(i)\downarrow= 1$ if $i \in W_e$ and $\Phi_i(i)\downarrow= 0$ if $i \in W_j$.[12] We then let $f(e,j) = i$.

Another example of a Π^0_1 class is the class of (codes of) extensions of a complete consistent theory. By the proof of Gödel's Incompleteness Theorem (see e.g. [139]), for a theory such as Peano Arithmetic (PA), the set A of sentences provable from PA and the set B of sentences refutable from PA form an effectively inseparable pair.

It is easy to check that the intersection of two Π^0_1 classes is again a Π^0_1 class, as is their union.

The complement of a Π^0_1 class is a Σ^0_1 *class*. Thus C is a Σ^0_1 class iff there is a computable relation R such that

$$C = \{A : \exists n\, R(A \restriction n)\}.$$

We can think of Σ^0_1 classes as the analogs for infinite sequences of c.e. sets of finite strings. Indeed, an important fact about Σ^0_1 classes is that they are exactly the subsets of 2^ω generated by c.e. sets of finite strings, in the following sense. (Here we use the notation of Section 1.2.)

[11]Sometimes a Π^0_1 class is defined to be the set of paths through a computable subtree T of $\omega^{<\omega}$. Such a class is *computably bounded* if there is a computable function f such that for each $\sigma \in T$, if $\sigma n \in T$ then $n \leqslant f(\sigma)$. The study of computably bounded Π^0_1 classes reduces to the study of the special case of Π^0_1 subclasses of 2^ω, since every computably bounded Π^0_1 class is computably equivalent to a Π^0_1 subclass of 2^ω. We will restrict our attention to such classes, so for us a Π^0_1 class will mean a Π^0_1 subclass of 2^ω.

[12]In greater detail: Let g be a computable function such that for each k, the machine $\Phi_{g(k)}$ ignores its input and waits until k enters either W_e or W_j, at which point the machine returns 0 if $k \in W_e$ and 1 if $k \in W_j$ (while if $k \notin W_e \cup W_j$ then the machine does not return). By the recursion theorem, there is an i such that $\Phi_i = \Phi_{g(i)}$. For this i, we have $\Phi_i(i)\downarrow= 1$ if $i \in W_e$ and $\Phi_i(i)\downarrow= 0$ if $i \in W_j$.

Definition 2.19.1. A set $A \subseteq 2^{<\omega}$ is *prefix-free* if it is an antichain with respect to the natural partial ordering of $2^{<\omega}$; that is, for all $\sigma \in A$ and all τ properly extending σ, we have $\tau \notin A$.

Prefix-free sets will play an important role in this book, for example in the definition of prefix-free Kolmogorov complexity given in Section 3.5. Clearly, a set $C \subseteq 2^\omega$ is open iff it is of the form $[\![A]\!]$ for some prefix-free $A \subseteq 2^{<\omega}$. The following result is the effective analog of this fact.

Proposition 2.19.2. *A set $C \subseteq 2^\omega$ is a Σ^0_1 class iff there is a c.e. set $W \subseteq 2^{<\omega}$ such that $C = [\![W]\!]$. This fact remains true if we require W to be prefix-free.*

Proof. Given a Σ^0_1 class C, let T be a computable tree such that $\overline{C} = [T]$. Let W be the set of minimal elements of $\overline{T}$, that is, strings $\sigma \notin T$ such that every proper substring of σ is in T. Then W is prefix-free and $C = [\![W]\!]$.

Conversely, let $W \subseteq 2^{<\omega}$ be a c.e. set. Then $[\![W]\!] = \{A : \exists\langle n, s\rangle\, (A \upharpoonright n \in W_s)\}$, so $[\![W]\!]$ is a Σ^0_1 class. □

Thus we can think of Σ^0_1 classes as effectively open sets, and of Π^0_1 classes as effectively closed sets. So when we have a Σ^0_1 class C, we assume that we have fixed a c.e. set $W \subseteq 2^{<\omega}$ such that $C = [\![W]\!]$ and let $C[s] = [\![W[s]]\!]$. Similarly, for the Π^0_1 class $P = \overline{C}$, we let $P[s] = \overline{C[s]}$.

It is a potentially confusing fact that Proposition 2.19.2 remains true if we require W to be computable, rather than merely c.e. The reason for this fact is that if $W \subset 2^{<\omega}$ is prefix-free and c.e., then

$$V = \{\sigma : \exists\tau \in W_{|\sigma|+1} \setminus W_{|\sigma|}\, (\tau \preccurlyeq \sigma)\}$$

is computable and prefix-free, and $[\![V]\!] = [\![W]\!]$.

We say that the Σ^0_1 classes $C_0, C_1, \ldots$ are *uniformly* Σ^0_1, and their complements are *uniformly* Π^0_1, if there are uniformly c.e. sets $V_0, V_1, \ldots \subseteq 2^{<\omega}$ such that $C_i = [\![V_i]\!]$ for all i. It is not hard to see that $P_0, P_1, \ldots$ are uniformly Π^0_1 classes iff there are uniformly computable trees $T_0, T_1, \ldots$ such that $P_i = [T_i]$ for all i.

As one might imagine, a Δ^0_1 *class* is one such that both it and its complement are Σ^0_1. It is not hard to see that C is a Δ^0_1 class iff $C = [\![F]\!]$ for some finite $F \subset 2^{<\omega}$, so a Δ^0_1 class is just a clopen subset of 2^ω. The classes $C_0, C_1, \ldots$ are *uniformly* Δ^0_1 if there are uniformly computable finite sets $F_0, F_1, \ldots \subset 2^{<\omega}$ such that $C_i = [\![F_i]\!]$ for all i.

We will not need all that much of the extensive theory of Π^0_1 and Σ^0_1 classes, but a few facts will be important, such as the following bits of folklore. For more on the subject, see for instance Cenzer [55] or Cenzer and Remmel [57].

A path P through a tree T is *isolated* if there is a $\sigma \prec P$ such that no other path through T extends σ.

Proposition 2.19.3. *Let T be a computable tree and let P be an isolated path through T. Then P is computable.*

Proof. Let $\sigma \prec P$ be such that no path through T other than P extends σ. For any $n > |\sigma|$, there is exactly one $\tau \succ \sigma$ such that the portion of T above τ is infinite (by König's Lemma, which says that if a finitely branching tree is infinite, then it has a path). So to compute $P \upharpoonright n$ for $n > |\sigma|$, we look for an $m \geqslant n$ such that exactly one $\tau \succ \sigma$ of length n has an extension of length m in T. Then $P \upharpoonright n = \tau$. □

Corollary 2.19.4. *Let C be a Π^0_1 class with only finitely many members. Then every member of C is computable.*

Proof. Let T be a computable tree such that $C = [T]$. Then T has only finitely many paths, so every path through T is isolated. □

Proposition 2.19.5. *Let C be a nonempty Π^0_1 class with no computable members. Then $|C| = 2^{\aleph_0}$.*

Proof. Let T be a computable tree such that $C = [T]$. Let S be the set of *extendible* elements of T, that is, those σ such that there is an extension of σ in $[T]$. If every $\sigma \in S$ has two incompatible extensions in S, then it is easy to show that there are continuum many paths through T. Otherwise, there is a $\sigma \in S$ with a unique extension $P \in [T]$. As in the previous proof, P is computable. □

Note also that the question of whether a given computable tree T is finite is an existential one, since it amounts to asking whether there is an n such that no string of length n is in T, and hence it can be decided by $\emptyset'$.

The *Cantor-Bendixson derivative* of a Π^0_1 class C is the class C' obtained from C by removing the isolated points of C (i.e., those elements $X \in C$ such that for some n, we have $[\![X \upharpoonright n]\!] \cap C = \{X\}$). Let $C^{(0)} = C$. For an ordinal δ, let $C^{(\delta+1)} = (C^{\delta})'$, and for a limit ordinal δ, let $C^{(\delta)} = \bigcap_{\gamma<\delta} C^{(\gamma)}$. The *rank* of $X \in C$ is the least δ such that $X \in C^{(\delta)} \setminus C^{(\delta+1)}$, if such a δ exists. Note that, by Proposition 2.19.3, if X has rank 0, then it is computable. If all the elements of C have ranks, then the *rank* of C is the supremum of the ranks of its elements (or, equivalently, the least δ such that $C^{(\delta+1)} = \emptyset$).

2.19.2 Π^0_n and Σ^0_n classes

There is a hierarchy of classes, akin to the arithmetic hierarchy of sets.

Definition 2.19.6. A set $C \subseteq 2^{\omega}$ is a Π^0_n *class* if there is a computable relation R such that

$$C = \{A : \forall k_1\, \exists k_2 \cdots Q k_n\, R(A \upharpoonright k_1, A \upharpoonright k_2, \ldots, A \upharpoonright k_n)\},$$

where the quantifiers alternate and hence $Q = \forall$ if n is odd and $Q = \exists$ if n is even.

The complement of a Π^0_n class is a Σ^0_n *class.*

These definitions can be relativized to a given set X (by letting R be X-computable) to obtain the notions of $\Pi^{0,X}_n$ class and $\Sigma^{0,X}_n$ class. A Π^0_{n+1} class is also a $\Pi^{0,\emptyset^{(n)}}_1$ class, since every predicate of the form $\exists k_2 \cdots Q k_n\, R(A \upharpoonright k_1, A \upharpoonright k_2, \dots, A \upharpoonright k_n)$ with R computable is computable in $\emptyset^{(n)}$. Similarly, every Σ^0_{n+1} class is also a $\Sigma^{0,\emptyset^{(n)}}_1$ class. The converse, however, is not always true, because every $\Pi^{0,\emptyset'}_1$ class is closed (indeed, every $\Pi^{0,X}_1$ class is closed for any X), but there are Π^0_2 classes that are not closed, such as the class C of all A with $A(m) = 1$ for infinitely many m, which can be written as $\{A : \forall k\, \exists m\, (m > k \wedge A(m) = 1)\}$.

Indeed, even a *closed* Π^0_2 class can fail to be a $\Pi^{0,\emptyset'}_1$ class. Doug Cenzer [personal communication] provided us the following example. It is not hard to build a computable subtree T of $\mathbb{N}^{<\omega}$ with a single infinite path $f \nleqslant_{\mathrm{T}} \emptyset'$. Let $\widehat{T} = \{1^{n_0+1}01^{n_1+1}0 \dots 1^{n_k+1}0^m : (n_0, \dots, n_k) \in T,\ m \in \mathbb{N}\}$. It is easy to check that $\widehat{T}$ is a computable subtree of $2^{<\omega}$. Let $P = [\widehat{T}] \cap C$, where C is as in the previous paragraph. Then P is a Π^0_2 class (because, as with Π^0_1 classes, the intersection of two Π^0_n classes is again a Π^0_n class, as is their union). Furthermore, P has a single member $A = 1^{f(0)+1}01^{f(1)+1}0 \dots$. But $A \equiv_{\mathrm{T}} f$, so $A \nleqslant_{\mathrm{T}} \emptyset'$. Thus P cannot be a $\Pi^{0,\emptyset'}_1$ class by the relativized form of Corollary 2.19.4.

Since $\Sigma^{0,\emptyset^{(n)}}_1$ *sets* are the same as Σ^0_n sets, the relativized form of Proposition 2.19.2 says that a class is $\Sigma^{0,\emptyset^{(n)}}_1$ iff it is of the form $[\![W]\!]$ for some Σ^0_n set $W \subseteq 2^{<\omega}$.

We fix effective listings $S^n_0, S^n_1, \dots$ of the Σ^0_n classes by letting $S^n_e = \{A : \exists k_1\, \forall k_2 \cdots Q k_n\, R(A \upharpoonright k_1, A \upharpoonright k_2, \dots, A \upharpoonright k_n)\}$ where e is a code for the n-ary computable relation R. By an *index* for a Σ^0_n class S we mean an i such that $S = S^n_i$. Similarly, by an index for a Π^0_n class P we mean an i such that $P = \overline{S^n_i}$. We do the same for relativized classes in the natural way. We then say that the classes $C_0, C_1, \dots$ are *uniformly* Σ^0_n if there is a computable function f such that $C_i = S^n_{f(i)}$ for all i, and *uniformly* Π^0_n if their complements are uniformly Σ^0_n. This definition agrees with the definition of uniformly Σ^0_1 classes given in the previous section because the proof of Proposition 2.19.2 is effective, as we now note.

Proposition 2.19.7. *From an index for a Σ^0_1 class C, we can computably find an index for a (prefix-free) c.e. set $W \subseteq 2^{<\omega}$ such that $C = [\![W]\!]$, and vice versa.*

Another way to look at this hierarchy of classes is in terms of effective unions and intersections. For example, in the same way that Π^0_1 and Σ^0_1 classes are the effective analogs of closed and open sets, respectively, Π^0_2

and Σ^0_2 classes are the effective analogs of G_δ and F_σ sets, respectively (where a set is G_δ if it is a countable intersection of open sets and F_σ if it is a countable union of closed sets). More precisely, C is a Π^0_2 class iff there are uniformly Σ^0_1 classes $U_0, U_1, \ldots$ such that $C = \bigcap_n U_n$. Similarly, C is a Σ^0_2 class iff there are uniformly Π^0_1 classes $P_0, P_1, \ldots$ such that $C = \bigcup_n P_n$. The same is true for any n in place of 1 and $n+1$ in place of 2. It is also easy to check that we can take $U_0 \supseteq U_1 \supseteq \cdots$ and $P_0 \subseteq P_1 \subseteq \cdots$ if we wish.

2.19.3 Basis theorems

A *basis* for the Π^0_1 classes is a collection of sets $\mathcal{C}$ such that every nonempty Π^0_1 class has an element in $\mathcal{C}$. One of the most important classes of results on Π^0_1 classes is that of basis theorems. A *basis theorem* for Π^0_1 classes states that every nonempty Π^0_1 class has a member of a certain type. (Henceforth, all Π^0_1 classes will be taken to be nonempty without further comment.) For instance, it is not hard to check that the lexicographically least element of a Π^0_1 class C (i.e., the leftmost path of a computable tree T such that $[T] = C$) has c.e. degree.[13] This fact establishes the Computably Enumerable Basis Theorem of Jockusch and Soare [195], which states that every Π^0_1 class has a member of c.e. degree, and is a refinement of the earlier Kreisel Basis Theorem, which states that every Π^0_1 class has a Δ^0_2 member.

The following is the most famous and widely applicable basis theorem.

Theorem 2.19.8 (Low Basis Theorem, Jockusch and Soare [196]). *Every Π^0_1 class has a low member.*

Proof. Let $C = [T]$ with T a computable tree. We define a sequence $T = T_0 \supseteq T_1 \supseteq \cdots$ of infinite computable trees such that if $P \in \bigcap_e [T_e]$ then P is low. (Note that there must be such a P, since each $[T_e]$ is closed.)

We begin with $T_0 = 2^{<\omega}$. Suppose that we have defined T_e, and let

$$U_e = \{\sigma : \Phi^\sigma_e(e)[|\sigma|]\uparrow\}.$$

Then U_e is a computable tree. If $U_e \cap T_e$ is infinite, then let $T_{e+1} = T_e \cap U_e$. Otherwise, let $T_{e+1} = T_e$. Note that either $\Phi^P_e(e)\uparrow$ for all $P \in [T_{e+1}]$ or $\Phi^P_e(e)\downarrow$ for all $P \in [T_{e+1}]$.

Now suppose that $P \in \bigcap_e [T_e]$. We can perform the above construction computably in $\emptyset'$, since $\emptyset'$ can decide the question of whether $U_e \cap T_e$ is finite. Thus $\emptyset'$ can decide whether $\Phi^P_e(e)\downarrow$ for any given e, and hence P is low. $\square$

The above proof has a consequence that will be useful below. Recall that a set A is superlow if $A' \equiv_{\text{tt}} \emptyset'$. Marcus Schaefer observed that the proof of the low basis theorem actually produces a superlow set.

[13]Such an element is in fact a *left-c.e. real*, a concept we will define in Chapter 5.

Corollary 2.19.9 (Jockusch and Soare [196], Schaefer). *Every Π^0_1 class has a superlow member.*

The following variation on the low basis theorem will be useful below.

Theorem 2.19.10 (Jockusch and Soare [196]). *Let C be a Π^0_1 class and let X be a noncomputable set. Then there is an $A \in C$ such that $X \nleqslant_{\mathrm{T}} A$ and $A' \leqslant_{\mathrm{T}} X \oplus \emptyset'$.*

Proof. Let $C = [T]$ with T a computable tree. As in the proof of the low basis theorem, we define a sequence $T = T_0 \supseteq T_1 \supseteq \cdots$ of infinite computable trees.

We begin with $T_0 = 2^{<\omega}$. Suppose that we have defined T_{2e} to be a computable tree. We first proceed as in the proof of the low basis theorem. Let $U_e = \{\sigma : \Phi^\sigma_e(e)[|\sigma|]\uparrow\}$. If $U_e \cap T_{2e}$ is infinite, then let $T_{2e+1} = T_{2e} \cap U_e$. Otherwise, let $T_{2e+1} = T_{2e}$. As before, T_{2e+1} is a computable tree.

We now handle cone-avoidance as follows. We $X \oplus \emptyset'$-computably search until we find that one of the following holds.

1. There are an n and an extendible $\sigma \in T_{2e+1}$ such that $\Phi^\sigma_e(n)\downarrow \neq X(n)$.
2. There is an n such that $T_{2e+1} \cap \{\sigma : \Phi^\sigma_e(n)\uparrow\}$ is infinite.

We claim one of the above must hold. Suppose not, and fix n. The fact that 2 does not hold means that we can find a k such that $\Phi^\sigma_e(n)\downarrow$ for all $\sigma \in T_{2e+1} \cap 2^k$. The fact that 1 does not hold means that, for each such σ, if $\Phi^\sigma_e(n) \neq X(n)$, then T_{2e+1} is finite above σ. Thus we can compute $X(n)$ by searching for a j such that for all $\sigma, \tau \in T_{2e+1} \cap 2^j$, we have $\Phi^\sigma_e(n)\downarrow = \Phi^\tau_e(n)\downarrow$, which by the above must exist. Since X is noncomputable, either 1 or 2 must hold. If we find that 1 holds, then let $T_{2e+2} = T_{2e+1} \cap \{\tau : \sigma \preccurlyeq \tau\}$. Otherwise, let $T_{2e+2} = T_{2e+1} \cap \{\sigma : \Phi^\sigma_e(n)\uparrow\}$.

Now let $A \in \bigcap_e [T_e]$. It is easy to check that we can perform the above construction computably in $X \oplus \emptyset'$, so as in the proof of the low basis theorem, we have $A' \leqslant_{\mathrm{T}} X \oplus \emptyset'$. Furthermore, the definition of the trees T_{2e+2} ensures that $X \nleqslant_{\mathrm{T}} A$. □

Corollary 2.19.9 and Theorem 2.19.10 cannot be combined. That is, there are a Π^0_1 class C and a noncomputable set X such that X is computable from every superlow member of C. We will see examples in Section 14.4.

Another important basis theorem is provided by the hyperimmune-free degrees.

Theorem 2.19.11 (Hyperimmune-Free Basis Theorem, Jockusch and Soare [196]). *Every Π^0_1 class has a member of hyperimmune-free degree.*

Proof. Let C be a Π^0_1 class, and let T be a computable tree such that $C = [T]$. We can carry out a construction like that in the proof of Miller and Martin's Theorem 2.17.3 within T. We begin with $T_0 = T$. We do not need the odd stages of that construction since we do not need to force noncomputability (as $\mathbf{0}$ is hyperimmune-free). Thus we deal with Φ_e at stage $e+1$.

We are given T_e and we want to build T_{e+1} to ensure that either

(i) if $A \in [T_{e+1}]$ then Φ^A_e is not total, or

(ii) there is a computable function f such that if $A \in [T_{e+1}]$ then $\Phi^A_e(n) \leqslant f(n)$ for all n.

Let

$$U^n_e = \{\sigma : \Phi^\sigma_e(n)\uparrow\}.$$

There are two cases.

If there is an n such that $U^n_e \cap T_e$ is infinite, then let $T_{e+1} = U^n_e \cap T_e$. In this case, if $A \in [T_{e+1}]$ then $\Phi^A_e(n)\uparrow$.

Otherwise, let $T_{e+1} = T_e$. In this case, for each n we can compute a number $l(n)$ such that no string $\sigma \in T_e$ of length $l(n)$ is in U^n_e. For any such σ, we have $\Phi^\sigma_e(n)\downarrow$. Define the computable function f by

$$f(n) = \max\{\Phi^\sigma_e(n) : \sigma \in T_e \wedge |\sigma| = l(n)\}.$$

Then $A \in [T_{e+1}]$ implies that $\Phi^A_e(n) \leqslant f(n)$ for all n.

So if we let $A \in \bigcap_e [T_e]$, then A is a member of C and has hyperimmune-free degree. $\square$

In the above construction, if $[T]$ has no computable members, then Proposition 2.19.5 implies that each T_e must have size $2^{\aleph_0}$. It is not hard to adjust the construction in this case to obtain continuum many members of C of hyperimmune-free degree. Thus, a Π^0_1 class with no computable members has continuum many members of hyperimmune-free degree.

The following result is an immediate consequence of the hyperimmune-free basis theorem and Theorem 2.17.2, but it was first explicitly articulated by Kautz [200], who gave an interesting direct proof. Recall that a set A is computably enumerable in and above a set X (CEA(X)) if A is c.e. in X and $X \leqslant_{\mathrm{T}} A$. A set A is *CEA* if it is CEA(X) for some $X <_{\mathrm{T}} A$.

Corollary 2.19.12 (Jockusch and Soare [196], Kautz [200]). *Every Π^0_1 class has a member that is not CEA.*

Although the existence of minimal degrees (Theorem 2.18.2) was proved by techniques similar to those used to prove the hyperimmune-free basis theorem, there is no "minimal basis theorem", since there are Π^0_1 classes with no members of minimal degree. An example is the Π^0_1 class of all complete extensions of Peano Arithmetic, by Theorem 2.21.9 below. However, using techniques beyond the scope of this book, Groszek and Slaman [173] proved the following elegant result.

Theorem 2.19.13 (Groszek and Slaman [173]). *There is a Π^0_1 class such that every member either is c.e. and noncomputable, or has minimal degree.*

We end this section by looking at examples of *nonbasis* theorems.

Proposition 2.19.14. *The intersection of all bases for the Π^0_1 classes is the collection of computable sets, and hence there is no minimal basis.*

Proof. If A is a computable set then $\{A\}$ is a Π^0_1 class, so every basis for the Π^0_1 classes must include A. On the other hand, if B is noncomputable then, by Corollary 2.19.4, $\{B\}$ is not a Π^0_1 class, so every Π^0_1 class must contain a member other than B, and hence the collection of all sets other than B is a basis for the Π^0_1 classes.[14] □

For our next result, we will need the following definition and lemma.

Definition 2.19.15. An infinite set A is *effectively immune* if there is a computable function f such that for all e, if $W_e \subseteq A$ then $|W_e| \leqslant f(e)$.

Post gave the following construction of an effectively immune co-c.e. set A. At stage s, for each $e < s$, if $W_e[s] \subseteq A_s$ and there is an $x \in W_e[s]$ such that $x > 2e$, then put the least such x into $\overline{A}$. Then A is infinite, and is effectively immune via the function $e \mapsto 2e$.

Lemma 2.19.16 (Martin [258])**.** *If a c.e. set B computes an effectively immune set, then B is Turing complete.*

Proof. Let B be c.e., and let $A \leqslant_{\mathrm{T}} B$ be effectively immune, as witnessed by the computable function f. Let Γ be a reduction such that $\Gamma^B = A$.

For each k we build a c.e. set $W_{h(k)}$, where the computable function h is given by the recursion theorem with parameters. Initially, $W_{h(k)}$ is empty. If k enters $\emptyset'$ at stage t then we wait until a stage $s \geqslant t$ at which there is a q such that $\Gamma^B(n)[s]\downarrow$ for all $n < q$ and there are more than $f(h(k))$ many numbers in $\Gamma^B[s] \upharpoonright q$. We then let $W_{h(k)} = \Gamma^B[s] \upharpoonright q$.[15] This action ensures that $|W_{h(k)}| > f(h(k))$, so we must have $W_{h(k)} \nsubseteq A$. Thus,

$$\Gamma^B[s] \upharpoonright q = W_{h(k)} \neq A \upharpoonright q = \Gamma^B \upharpoonright q.$$

So to compute $\emptyset'(k)$ from B, we simply look for a stage s at which there is a q such that $B_s \upharpoonright \gamma^B(n)[s] = B \upharpoonright \gamma^B(n)[s]$ for all $n < q$ and there are more than $f(h(k))$ many numbers in $\Gamma^B[s] \upharpoonright q$. If k is not in $\emptyset'$ by stage s, it cannot later enter $\emptyset'$. □

[14] Another way to prove this fact is to note that, by Theorem 2.17.2, the only sets that are both low and hyperimmune-free are the computable ones.

[15] In greater detail: For each i and k, we define a c.e. set $C_{i,k}$ as follows. Initially, $C_{i,k}$ is empty. If k enters $\emptyset'$ at stage t then we wait until a stage $s \geqslant t$ at which there is a q such that $\Gamma^B(n)[s]\downarrow$ for all $n < q$ and there are more than $f(i)$ many numbers in $\Gamma^B[s] \upharpoonright q$. We then let $C_{i,k} = \Gamma^B[s] \upharpoonright q$. It is easy to see that there is a computable function g such that $g(i,k)$ is an index for $C_{i,k}$ for all i and k. By the recursion theorem with parameters, there is a computable function h such that $W_{h(k)} = W_{g(h(k),k)}$ for all k. This h has the required properties.

Theorem 2.19.17 (Jockusch and Soare [195]). (i) *The incomplete c.e. degrees do not form a basis for the* Π^0_1 *classes.*

(ii) *Let* $\mathbf{a}$ *be an incomplete c.e. degree. Then the degrees less than or equal to* $\mathbf{a}$ *do not form a basis for the* Π^0_1 *classes.*

Proof. By Lemma 2.19.16, to prove both parts of the theorem, it is enough to find a Π^0_1 class whose members are all effectively immune. Let A be Post's effectively immune set described after Definition 2.19.15. An infinite subset of an effectively immune set is clearly also effectively immune, so it is enough to find a Π^0_1 class all of whose members are infinite subsets of A.

It is clear from the construction of A that $|\overline{A} \upharpoonright 2e| \leqslant e$ for all e. Thus $A \cap [2^k - 1, 2^{k+1} - 2] \neq \emptyset$ for all k.[16] Let

$$\mathcal{P} = \{B : B \subseteq A \wedge \forall k\, (B \cap [2^k - 1, 2^{k+1} - 2] \neq \emptyset)\}.$$

Since A is co-c.e., $\mathcal{P}$ is a Π^0_1 class, and it is nonempty because $A \in \mathcal{P}$. Furthermore, every element of $\mathcal{P}$ is an infinite subset of A. As explained in the previous paragraph, these facts suffice to establish the theorem. □

2.19.4 Generalizing the low basis theorem

We would like to extend the low basis theorem to degrees above $\mathbf{0}'$. Of course, we cannot hope to prove that for every degree $\mathbf{a} \geqslant \mathbf{0}'$, every Π^0_1 class has a member whose jump has degree $\mathbf{a}$, since there are Π^0_1 classes whose members are all computable. We can prove this result, however, if we restrict our attention to *special* Π^0_1 classes, which are those with no computable members.

Theorem 2.19.18 (Jockusch and Soare [196]). *If* C *is a* Π^0_1 *class with no computable members and* $\mathbf{a} \geqslant \mathbf{0}'$*, then there is a* $P \in C$ *such that* $\deg(P') = \deg P \oplus \emptyset' = \mathbf{a}$.

Proof. Let $C = [T]$ with T a computable tree and let $A \in \mathbf{a}$. By Proposition 2.19.5, T has $2^{\aleph_0}$ many paths. We A-computably define an infinite sequence of computable trees $T_0 \supseteq T_1 \supseteq \cdots$, each with $2^{\aleph_0}$ many paths. We begin with $T_0 = T$.

Given T_{2e}, let

$$U_{2e} = \{\sigma \in T_{2e} : \Phi^\sigma_e(e)[|\sigma|]\uparrow\}.$$

If U_{2e} is finite then let $T_{2e+1} = T_{2e}$. Otherwise, since U_{2e} is a subtree of T, it has no computable paths, and hence has $2^{\aleph_0}$ many paths. In this case, let $T_{2e+1} = U_{2e}$. Notice that we can A-computably decide which case we are in because $A \geqslant_{\mathrm{T}} \emptyset'$.

[16]Notice that this fact shows that A is not hyperimmune. In fact, it is not hard to show that a co-c.e. set X is not hyperimmune iff there is a Π^0_1 class all of whose members are infinite subsets of X.

Now let $\sigma_0 <_{\mathrm{L}} \sigma_1$ be the length-lexicographically least pair of incomparable strings in T_{2e+1} such that for each $i < 2$, the subtree of T_{2e+1} above σ_i is infinite (and hence has $2^{\aleph_0}$ many paths). Notice that we can find σ_0, σ_1 using A, again because $A \geqslant_{\mathrm{T}} \emptyset'$. Let T_{2e+2} be the subtree of T_{2e+1} above $\sigma_{A(e)}$.

Having defined the T_e, let $P \in \bigcap_e [T_e]$. The choice of T_{2e+1} determines whether $\Phi_e^P(e)\downarrow$, so, since the entire construction is A-computable, $P' \leqslant_{\mathrm{T}} A$. On the other hand, we can also perform the construction $(P \oplus \emptyset')$-computably, since all the steps can be done $\emptyset'$-computably, except the choice of which σ_i to extend in defining T_{2e+2}, which can be done P-computably, since the σ_i we choose at that stage is the one extended by P. Furthermore, that choice determines the eth bit of A. Thus $A \leqslant_{\mathrm{T}} P \oplus \emptyset' \leqslant_{\mathrm{T}} P'$, and hence $P' \equiv_{\mathrm{T}} P \oplus \emptyset' \equiv_{\mathrm{T}} A$. ☐

Corollary 2.19.19 (Jockusch and Soare [196]). *For each* $\mathbf{a} \geqslant \mathbf{0}'$*, the sets with degrees in* $\{\mathbf{b} : \mathbf{b}' = \mathbf{a}\} \cup \{\mathbf{0}\}$ *form a basis for the* Π_1^0 *classes.*

2.20 Strong reducibilities and Post's Program

The concept of hypersimplicity, which we defined in Section 2.17, was introduced by Post [316] as part of an attempt to solve Post's Problem. Although hypersimple sets can be Turing complete, Post [316] showed that no hypersimple set can be tt-complete. Later, Friedberg and Rogers [162] proved that no hypersimple set can be wtt-complete. This result has been extended as follows. A set A is *wtt-cuppable* if there is a c.e. set $W <_{\mathrm{wtt}} \emptyset'$ such that $A \oplus W \geqslant_{\mathrm{wtt}} \emptyset'$.

Theorem 2.20.1 (Downey and Jockusch [119]). *No hypersimple set is wtt-cuppable.*

Proof. Suppose that A is hypersimple, W is c.e., $W \leqslant_{\mathrm{wtt}} \emptyset'$, and $A \oplus W \geqslant_{\mathrm{wtt}} \emptyset'$. We show that $W \geqslant_{\mathrm{wtt}} \emptyset'$. We will build a c.e. set B. By the recursion theorem, we may assume that we have a wtt-reduction $\Gamma^{A \oplus W} = B$ with computable use g.[17] Without loss of generality, we may assume that $g(n)$ is increasing and even for all n and write $h(n)$ for $\frac{g(n)}{2}$.

We define a strong array $\{F_i\}_{i \in \mathbb{N}}$ as follows, writing $m(i)$ for $\max F_i$. Let $F_0 = [0, h(0)]$. Given F_i, pick $m(i) + 1$ many fresh large numbers $b_0^{i+1} < \cdots < b_{m(i)}^{i+1}$ and let $F_{i+1} = (\max F_i, h(b_{m(i)}^{i+1})]$. Since the F_i form a strong array, there are infinitely many i with $F_i \subseteq A$. We can therefore

[17]In greater detail: We can think of the ensuing construction as building a c.e. set $W_{f(e)}$ given a parameter e, taking as Γ a wtt-functional such that $\Gamma^{A \oplus W} = W_e$. (It is not hard to see that indices for Γ and for a computable function bounding its use can be found effectively from e.) By the recursion theorem, there is an e such that $W_e = W_{f(e)}$, and we let $B = W_e$.

enumerate an increasing computable sequence $i_0 < i_1 < \cdots$ such that $F_{i_k+1} \subseteq A$ for all k.

Now for each k, we act as follows. Let $b = b^{i_k+1}_{m(i_k)}$. We wait until a stage s_k such that $F_{i_k+1} \subseteq A_{s_k}$. If k enters $\emptyset'$ at a stage $t_k \geqslant s_k$, we wait for a stage s such that $\Gamma^{A\oplus W}[s] \upharpoonright (b+1) = B_s \upharpoonright (b+1)$, then put $b^{i_k+1}_0$ into B. If later A changes below $h(b)$, we again wait for a stage s such that $\Gamma^{A\oplus W}[s] \upharpoonright (b+1) = B_s \upharpoonright (b+1)$, then put $b^{i_k+1}_1$ into B. We continue in this manner, enumerating one more $b^{i_k+1}_j$ into B each time A changes below $h(b)$ and the computation $\Gamma^{A\oplus W} \upharpoonright (b+1)$ recovers. Since $F_{i_k+1} \subseteq A_{s_k}$, we cannot have more than $m(i_k)+1$ many changes in A below $h(b)$ after stage s_k, so we never run out of $b^{i_k+1}_j$'s. Since we change $B \upharpoonright (b+1)$ after the last time $A \upharpoonright h(b)$ changes, $W \upharpoonright h(b)$ must change at that point to ensure that $\Gamma^{A\oplus W} \upharpoonright (b+1) = B \upharpoonright (b+1)$.

Thus $W \upharpoonright h(b)$ changes at least once after stage t_k, so we can wtt-compute $\emptyset'$ using W as follows. Given k, wait for a stage $s \geqslant s_k$ such that $W_s \upharpoonright h(b) = W \upharpoonright h(b)$. Then $k \in \emptyset'$ iff $k \in \emptyset'_s$. □

Post's original program for solving Post's Problem was to find a "thinness" property of the lattice of supersets of a c.e. set (under inclusion) guaranteeing Turing incompleteness of the given set. It was eventually shown that this approach could not succeed. Although Sacks [343] constructed a *maximal set* (i.e., a coinfinite c.e. set M such that if $W \supseteq M$ is c.e. then either W is cofinite or $W \setminus M$ is finite) that is Turing incomplete, it is also possible to construct Turing complete maximal sets. Indeed, Soare [365] showed that the maximal sets form an orbit of the automorphisms of the lattice of c.e. sets under inclusion, and hence there is no definable property that can be added to maximality to guarantee incompleteness. Eventually, Cholak, Downey, and Stob [66] showed that no property of the lattice of c.e. supersets of a c.e. set can by itself guarantee incompleteness. Finally, Harrington and Soare [176] did find an elementary property of c.e. sets (that is, one that is first-order definable in the language of the lattice of c.e. sets) that guarantees incompleteness. There is a large amount of fascinating material related to this topic. For instance, Cholak and Harrington [68] have shown that one can define all "double jump" classes in the lattice of c.e. sets using infinitary formulas. The methods are intricate and rely on analyzing the failure of the "automorphism machinery" first developed by Soare and later refined by himself and others, particularly Cholak and Harrington.

Properties like simplicity (defined in Section 2.2) and hypersimplicity do have implications for the degrees of sets related to a given c.e. set. For instance, Stob [383] showed that a c.e. set is simple iff it does not have c.e. supersets of all c.e. degrees. Downey [100] showed that if A is hypersimple then there is a c.e. degree $\mathbf{b} \leqslant \deg(A)$ such that if $A_0 \sqcup A_1$ is a c.e. splitting of A, then neither of the A_i has degree $\mathbf{b}$.

2.21 PA degrees

In this section, we assume familiarity with basic concepts of mathematical logic, such as the notion of a first-order theory, the formal system of Peano Arithmetic, and Gödel's Incompleteness Theorem, as may be found in an introductory text such as Enderton [139]. Those unfamiliar with this material may take Corollary 2.21.4 below as a definition of the notion of a PA degree (which makes Theorem 2.21.3 immediate).

The class of complete extensions of a given computably axiomatizable first-order theory is a Π^0_1 class. (We assume here that all theories are consistent.) Jockusch and Soare [195, 196] showed that, in fact, for every Π^0_1 class P there is a computably axiomatizable first-order theory T such that the class of degrees of members of P coincides with the class of degrees of complete extensions of T. (This result was later extended by Hanf [174] to finitely axiomatizable theories.) As we will see, there are Π^0_1 classes P that are *universal*, in the sense that any set that can compute an element of P can compute an element of any nonempty Π^0_1 class. A natural example of such a class is the class of complete extensions of Peano Arithmetic. The degrees of elements of this class are known as *PA degrees.*

Definition 2.21.1. A degree $\mathbf{a}$ is a *PA degree* if it is the degree of a complete extension of Peano Arithmetic.

Theorem 2.21.2 (Scott Basis Theorem [352]). *If* $\mathbf{a}$ *is a PA degree then the sets computable from* $\mathbf{a}$ *form a basis for the* Π^0_1 *classes.*

Proof. Let S be a complete extension of PA of degree $\mathbf{a}$.[18] Let T be an infinite computable tree. We compute a path through T using S by defining a sequence $\sigma_0 \prec \sigma_1 \prec \cdots \in T$.

Let $\sigma_0 = \lambda$. Suppose that σ_n has been defined in such a way that there is a path through T extending σ_n. If there is a unique i such that $\sigma_n i \in T$, then let $\sigma_{n+1} = \sigma_n i$. Otherwise, both $\sigma_n 0$ and $\sigma_n 1$ are in T. For $i = 0, 1$, let θ_i be the sentence

$$\exists m\,(\sigma_n i \text{ has an extension of length } m \text{ in } T \text{ but } \sigma_n(1-i) \text{ does not}).$$

These sentences can be expressed in the language of first-order arithmetic, by the kind of coding used in the proof of Gödel's Incompleteness Theorem.

Within PA, we can argue as follows. Suppose that θ_0 and θ_1 both hold. Then for each $i = 0, 1$, there is a least m_i such that $\sigma_n i$ has an extension of length m_i in T but $\sigma_n(1-i)$ does not. Let i be such that $m_i \geqslant m_{1-i}$. Then $\sigma_n(1-i)$ has an extension of length m_{1-i}, and hence one of length m_i, which contradicts the choice of m_i.

[18] We do not actually need S to be complete, but only consistent and deductively closed.

Thus PA $\vDash \neg(\theta_0 \wedge \theta_1)$, so there is a least i such that $\theta_{1-i} \notin S$. Let $\sigma_{n+1} = \sigma_n i$. There must be a path through T extending σ_{n+1}, since otherwise we would have PA $\vDash \theta_{1-i}$, and hence $\theta_{1-i} \in S$. □

In unpublished work, Solovay showed that, in fact, the property in Theorem 2.21.2 exactly characterizes the PA degrees. This fact, which we will state as Corollary 2.21.4 below, follows immediately from the following result, which is important in its own right.

Theorem 2.21.3 (Solovay [unpublished]). *The class of PA degrees is closed upwards.*

Proof. Let A be the set of (Gödel numbers of) sentences provable in PA and B the set of (Gödel numbers of) sentences refutable from PA. We follow the proof in Odifreddi [310], which uses the fact that A and B form an effectively inseparable pair of c.e. sets (which recall means that there is a computable function f such that for all disjoint c.e. $W_e \supseteq A$ and $W_i \supseteq B$, we have $f(e,i) \notin W_e \cup W_i$).

Let T be a complete extension of PA computable in a set X. Using T, we will build finite extensions F_σ of PA for $\sigma \in 2^{<\omega}$ so that $\bigcup_{\sigma \prec X} F_\sigma$ is a complete extension of PA with the same degree as X.

Let $\varphi_0, \varphi_1, \dots$ be an effective listing of the sentences of arithmetic. Let $F_\lambda =$ PA. Suppose that we have defined F_σ to be a finite consistent extension of PA. Let ψ_0 be an existential sentence of arithmetic expressing the statement that

$$\exists m\,(m \text{ codes a proof of } \varphi_n \text{ in } F_\sigma \text{ and no } k < m \text{ codes a proof of } \neg\varphi_n \text{ in } F_\sigma).$$

Let ψ_1 be the same with the roles of φ_n and $\neg\varphi_n$ interchanged.

We have $\psi_i \in T$ iff ψ_i is true, and at most one ψ_i is true. If $\psi_0 \in T$ then let $F'_\sigma = F_\sigma \cup \{\varphi_n\}$ and otherwise let $F'_\sigma = F_\sigma \cup \{\neg\varphi_n\}$.

Let C be the sentences provable in F'_σ and D the sentences refutable from F'_σ. Then C and D are c.e. sets containing A and B, respectively, and we can effectively find indices for C and D. Since A and B form an effectively inseparable pair, we can compute a sentence $\nu \notin C \cup D$. Let $F_{\sigma 0} = F'_\sigma \cup \{\nu\}$ and $F_{\sigma 1} = F'_\sigma \cup \{\neg\nu\}$.

Let $S = \bigcup_{\sigma \prec X} F_\sigma$. Then S is a complete extension of PA, and is X-computable, since T is X-computable. Given S, we can compute X as follows. Assume by induction that we have computed $X \upharpoonright n$ and $F_{X \upharpoonright n}$. We know which of φ_n and $\neg\varphi_n$ is in S, so we know $F'_{X\upharpoonright n}$. Thus we can compute ν. We know which of ν and $\neg\nu$ is in S, so we know $X \upharpoonright n+1$ and $F_{X\upharpoonright n+1}$. Thus $S \equiv_{\mathrm{T}} X$. □

Since the complete extensions of PA form a Π^0_1 class, if the sets computable from a degree **a** form a basis for the Π^0_1 classes, then in particular

a computes a complete extension of PA, and hence, by the above theorem, is itself a PA degree. Thus we have the following result.

Corollary 2.21.4 (Solovay). *A degree is PA iff the sets computable from it form a basis for the Π^0_1 classes.*

A useful way to restate this corollary is that a degree **a** is PA iff every computable infinite subtree of $2^{<\omega}$ has an **a**-computable path.

Another way to characterize PA degrees is via effectively inseparable pairs of c.e. sets. Let A and B form such a pair. As previously mentioned, the class of all sets that separate A and B is a Π^0_1 class, so every PA degree computes such a set. Conversely, given a set C separating A and B, we can use the following useful lemma to obtain a C-computable complete extension of PA.

Lemma 2.21.5 (Muchnik [285], Smullyan [362]). *Let C be a separating set for an effectively inseparable pair of c.e. sets A and B, and let X and Y be disjoint c.e. sets. Then there is a C-computable set separating X and Y.*

Proof. Let f be a total function witnessing the effective inseparability of A and B. It is not hard to see that we may assume that f is 1-1. Using the recursion theorem with parameters, we can define computable 1-1 functions g and h such that for all n we have

$$W_{g(n)} = \begin{cases} A \cup \{f(g(n), h(n))\} & \text{if } n \in Y \\ A & \text{otherwise} \end{cases}$$

and

$$W_{h(n)} = \begin{cases} B \cup \{f(g(n), h(n))\} & \text{if } n \in X \\ B & \text{otherwise.}^{19} \end{cases}$$

Let $D = \{n : f(g(n), h(n)) \in C\}$. If $n \in X$ then $f(g(n), h(n)) \in W_{g(n)} \cup W_{h(n)}$, so $W_{g(n)}$ and $W_{h(n)}$ cannot be disjoint. It follows that $f(g(n), h(n)) \in A \subseteq C$, so $n \in D$. The same argument shows that if $n \in Y$ then $f(g(n), h(n)) \in B$, so $n \notin D$. Thus D is a separating set for X and Y. □

Theorem 2.21.6 (Jockusch and Soare [196]). *If C is a separating set for an effectively inseparable pair then C has PA degree. Thus a degree is PA iff it computes a separating set for an effectively inseparable pair.*

Proof. Let X be the set of sentences provable in PA and Y the set of sentences refutable from PA. By the above lemma, there is a C-computable set D separating X and Y. Let $\varphi_0, \varphi_1, \ldots$ be an effective listing of the

[19] It is a useful exercise to work out the full details of the application of the recursion theorem in this case.

sentences of arithmetic. Let $T_0 = \emptyset$. Suppose we have defined the finite set of sentences T_n. Let $\psi = (\bigwedge_{\varphi \in T_n} \varphi) \rightarrow \varphi_n$. If $\psi \in D$ then we know ψ is not refutable from PA, so $(\bigwedge_{\varphi \in T_n} \varphi) \rightarrow \neg\varphi_n$ is not provable from PA, and we can let $T_{n+1} = T_n \cup \{\varphi_n\}$. Otherwise, ψ is not provable from PA, so we can let $T_{n+1} = T_n \cup \{\neg\varphi_n\}$. Let $T = \bigcup_n T_n$. It is easy to check that T is a D-computable (and hence C-computable) complete extension of PA. $\square$

The following result follows immediately from the Low Basis Theorem 2.19.8. (Indeed, this result was Jockusch and Soare's first application of their theorem.)

Theorem 2.21.7 (Jockusch and Soare [196]). *There is a low PA degree.*

By upward closure, $\mathbf{0}'$ is a PA degree. However, combining Theorems 2.19.17 and 2.21.2, we see that it is the only c.e. PA degree.

Theorem 2.21.8 (Jockusch and Soare [195]). *No incomplete c.e. degree is PA.*

The following is another useful fact about the distribution of PA degrees.

Theorem 2.21.9 (Scott and Tennenbaum [351]). *A PA degree cannot be minimal.*

Proof. Let $P = \{A \oplus B : \forall e\,(A(e) \neq \Phi_e(e) \wedge B(e) \neq \Phi_e^A(e))\}$. If a set S has PA degree then there is an $A \oplus B \in P$ such that $A \oplus B \leqslant_{\mathrm{T}} S$. It follows from the definition of P that $0 <_{\mathrm{T}} A <_{\mathrm{T}} A \oplus B \leqslant_{\mathrm{T}} S$, so S does not have minimal degree. $\square$

2.22 Fixed-point free and diagonally noncomputable functions

A total function f is *fixed-point free* if $W_{f(e)} \neq W_e$ for all e. By the recursion theorem, no computable function is fixed-point free, and indeed the fixed-point free functions can be thought of as those that avoid having fixed points in the sense of the recursion theorem. As we will see (in Section 8.17, for example), fixed-point free functions have interesting ramifications in both classical computability theory and the theory of algorithmic randomness. A related concept is that of a *diagonally noncomputable* (*DNC*) function, where the total function g is DNC if $g(e) \neq \Phi_e(e)$ for all e. A degree is diagonally noncomputable if it computes a DNC function.

Theorem 2.22.1 (Jockusch, Lerman, Soare, and Solovay [191]). *The following are equivalent.*

(i) *The set A computes a fixed-point free function.*

(ii) *The set A computes a total function h such that $\Phi_{h(e)} \neq \Phi_e$ for all e.*

(iii) *The set A computes a DNC function.*

(iv) *For each e there is a total function $h \leqslant_{\mathrm{T}} A$ such that $h(n) \neq \Phi_e(n)$ for all n.*

(v) *For each total computable function f there is an index i with Φ_i^A total and $\Phi_i^A(n) \neq \Phi_{f(i)}(n)$ for all n.*

Proof. We begin by proving the equivalence of (i)–(iii).

Clearly (i) implies (ii), since if $W_{h(e)} \neq W_e$ then $\Phi_{h(e)} \neq \Phi_e$.

To show that (ii) implies (iii), suppose that h is as in (ii). Define $\Phi_{d(u)}$ as in the proof of the recursion theorem. That is, $\Phi_{d(u)}(z)$ is $\Phi_{\Phi_u(u)}(z)$ if $\Phi_u(u)\downarrow$, and $\Phi_{d(u)}(z)\uparrow$ otherwise. As we have seen, we can choose d to be a total computable function. Let $g = h \circ d$. Note that $g \leqslant_{\mathrm{T}} A$. Suppose that $g(e) = \Phi_e(e)$. Then by (ii) we have

$$\Phi_{d(e)} \neq \Phi_{h(d(e))} = \Phi_{g(e)} = \Phi_{\Phi_e(e)} = \Phi_{d(e)},$$

which is a contradiction. So g is DNC.

To show that (iii) implies (i), fix a partial computable function ψ such that, for all e, if $W_e \neq \emptyset$ then $\psi(e) \in W_e$, and let q be partial computable with $\Phi_{q(e)}(q(e)) = \psi(e)$ for all e. Let $g \leqslant_{\mathrm{T}} A$ be DNC, and define $f \leqslant_{\mathrm{T}} A$ so that $W_{f(e)} = \{g(q(e))\}$. Suppose that $W_e = W_{f(e)}$. Then $W_e \neq \emptyset$, so $\psi(e) \in W_e$, which implies that $\psi(e) = g(q(e))$. But g is DNC, so $g(q(e)) \neq \Phi_{q(e)}(q(e)) = \psi(e)$, which is a contradiction. So f is fixed-point free.

We now prove the equivalence of (iv) and (v).

To show that (iv) implies (v), let f be a computable function, and let e be such that $\Phi_e(\langle i, j\rangle) = \Phi_{f(i)}(j)$. Fix $h \leqslant_{\mathrm{T}} A$ satisfying (iv) for this e. For all i and n, let $h_i(n) = h(\langle i, n\rangle)$. Then there is a total computable function p with $\Phi^A_{p(i)} = h_i$ for all i. By the relativized form of the recursion theorem, there is an i such that $\Phi^A_{p(i)} = \Phi^A_i$. Then Φ^A_i is total, and

$$\Phi_i^A(n) = \Phi^A_{p(i)}(n) = h(\langle i, n\rangle) \neq \Phi_e(\langle i, n\rangle) = \Phi_{f(i)}(n).$$

To show that (v) implies (iv), assume that (iv) fails for some e. Let $f(i) = e$ for all i. Then for all i such that Φ_i^A is total, there is an n such that $\Phi_i^A(n) = \Phi_e(n) = \Phi_{f(i)}(n)$, so (v) fails.

Finally, we show that (iii) and (iv) are equivalent.

To show that (iii) implies (iv), given e, let f be a total computable function such that $\Phi_{f(n)}(x) = \Phi_e(n)$ for all n and x. Let $g \leqslant_{\mathrm{T}} A$ be DNC and let $h = g \circ f$. Then

$$\Phi_e(n) = \Phi_{f(n)}(f(n)) \neq g(f(n)) = h(n).$$

To show that (iv) implies (iii), let e be such that $\Phi_e(n) = \Phi_n(n)$ for all n, and let h be as in (iv). Then $h(n) \neq \Phi_e(n) = \Phi_n(n)$ for all n, so h is DNC. $\square$

Further results about DNC functions, including ones relating them to Kolmogorov complexity, will be discussed in Chapter 8, particularly in Theorem 8.16.8.

For DNC functions with range $\{0,1\}$, we have the following result.

Theorem 2.22.2 (Jockusch and Soare [195], Solovay [unpublished]). *A degree is PA iff it computes a $\{0,1\}$-valued DNC function.*

Proof. It is straightforward to define a Π^0_1 class $\mathcal{P}$ such that the characteristic function of any element of $\mathcal{P}$ is a $\{0,1\}$-valued DNC function. So by the Scott Basis Theorem 2.21.2, every PA degree computes a $\{0,1\}$-valued DNC function.

Now let A compute a $\{0,1\}$-valued DNC function g. Then g is the characteristic function of a set separating the effectively inseparable pair $\{e : \Phi_e(e) = 0\}$ and $\{e : \Phi_e(e) = 1\}$. So by Lemma 2.21.6, g has PA degree. Since the class of PA degrees is closed upwards, so does A. $\square$

The following is a classic result on the interaction between fixed-point free functions and computable enumerability.

Theorem 2.22.3 (Arslanov's Completeness Criterion [13]). *A c.e. set is Turing complete iff it computes a fixed-point free function.*

Proof. It is easy to define a total $\emptyset'$-computable fixed-point free function. For the nontrivial implication, let A be a c.e. set that computes a fixed-point free function f. By speeding up the enumeration of A, we may assume we have a reduction $\Gamma^A = f$ such that $\Gamma^A(n)[s]\downarrow$ for all s and $n \leqslant s$.

For each n we build a set $W_{h(n)}$, with the total computable function h given by the recursion theorem with parameters. Initially, $W_{h(n)} = \emptyset$. If $n \in \emptyset'_s$ then for every $x \in W_{\Gamma^A(h(n))[s]}[s]$, we put x into $W_{h(n)}$ if it is not already there. (The full details of the application of the recursion theorem are much as in the footnote to the proof of Theorem 2.19.16.)

We claim we can compute $\emptyset'$ from A using h. To prove this claim, fix n. If $n \in \emptyset'$, then consider the stage s at which n enters $\emptyset'$. If the computation $\Gamma^A(h(n))$ has settled by stage s, then every $x \in W_{\Gamma^A(h(n))}$ is eventually put into $W_{h(n)}$, while no other x ever enters $W_{h(n)}$. So $W_{f(h(n))} = W_{\Gamma^A(h(n))} = W_{h(n)}$, contradicting the choice of f. Thus it must be the case that the computation $\Gamma^A(h(n))$ has not settled by stage s.

So, to decide whether $n \in \emptyset'$, we simply look for a stage s by which the computation $\Gamma^A(h(n))$ has settled (which we can do computably in A because A is c.e.). Then $n \in \emptyset'$ iff $n \in \emptyset'_s$. $\square$

The above argument clearly works for wtt-reducibility as well, so a c.e. set is wtt-complete iff it wtt-computes a fixed-point free function. Arslanov [14] has investigated similar completeness criteria for other reducibilities such as tt-reducibility.

Antonin Kučera realized that fixed-point free functions have much to say about c.e. degrees, even beyond Arslanov's Completeness Criterion, and, as we will later see, about algorithmic randomness. In particular, in [216] he showed that if $f \leqslant_{\mathrm{T}} \emptyset'$ is fixed-point free, then there is a noncomputable c.e. set $B \leqslant_{\mathrm{T}} f$. We will prove this result in a slightly stronger form after introducing the following notion.

Definition 2.22.4 (Maass [256]). A coinfinite c.e. set A is *promptly simple* if there are a computable function f and an enumeration $\{A_s\}_{s\in\omega}$ of A such that for all e,

$$|W_e| = \infty \;\Rightarrow\; \exists^\infty x\, \exists s\, (x \in W_e[s] \setminus W_e[s-1] \;\wedge\; x \in A_{f(s)}).$$

The idea of this definition is that x needs to enter A "promptly" after it enters W_e. We will say that a degree is promptly simple if it contains a promptly simple set. The usual simple set constructions (such as that of Post's hypersimple set) yield promptly simple sets.

We can now state the stronger form of Kučera's result mentioned above.

Theorem 2.22.5 (Kučera [216]). *If $f \leqslant_{\mathrm{T}} \emptyset'$ is fixed-point free then there is a promptly simple c.e. set $B \leqslant_{\mathrm{T}} f$.*

Proof. Let $f \leqslant_{\mathrm{T}} \emptyset'$ be fixed-point free. We build $B \leqslant_{\mathrm{T}} f$ to satisfy the following requirements for all e:

$$\mathcal{R}_e : |W_e| = \infty \;\Rightarrow\; \exists x\, \exists s\, (x \in W_e[s] \setminus W_e[s-1] \;\wedge\; x \in B_s).$$

To see that these requirements suffice to ensure the prompt simplicity of B (assuming that we also make B c.e. and coinfinite), notice that it is not hard to define a computable function g such that for all e and sufficiently large n, there is an i such that $W_i = W_e \cap \{n, n+1, \ldots\}$ and for $x \geqslant n$, if $x \in W_e[s] \setminus W_e[s-1]$ then $x \in W_i[g(s)] \setminus W_i[g(s)-1]$. So if the requirements are satisfied, then for each e and n,

$$|W_e| = \infty \;\Rightarrow\; \exists x > n\, \exists s\, (x \in W_e[s] \setminus W_e[s-1] \;\wedge\; x \in B_{g(s)}).$$

During the construction, we will define an auxiliary computable function h using the recursion theorem with parameters. (We will give the details of this somewhat subtle application of the recursion theorem after outlining the construction.)

At stage s, act as follows for each $e \leqslant s$ such that

1. $\mathcal{R}_e$ is not yet met,
2. some $x > \max\{2e, h(e)\}$ is in $W_e[s] \setminus W_e[s-1]$, and
3. $f_s(h(e)) = f_t(h(e))$ for all t with $x \leqslant t \leqslant s$.

Enumerate x into B (which causes $\mathcal{R}_e$ to be met). Using the recursion theorem with parameters, let $W_{h(e)} = W_{f_s(h(e))}$.

In greater detail, we think of the construction as having a parameter i and enumerating a set B_i. We define an auxiliary partial computable function g, which we then use to define h.

At stage s, act as follows for each $e \leqslant s$ such that

1. $\mathcal{R}_e$ is not yet met (with B_i in place of B),
2. some $x > \max\{2e, i\}$ is in $W_e[s] \setminus W_e[s-1]$, and
3. $f_s(i) = f_t(i)$ for all t with $x \leqslant t \leqslant s$.

Enumerate x into B_i (which causes $\mathcal{R}_e$ to be met) and let $g(i,e) = f_s(i)$. Using the recursion theorem with parameters (in the version for partial computable functions mentioned following the proof of Theorem 2.3.2), let h be a total computable function such that $W_{h(e)} = W_{g(h(e),e)}$ whenever $g(h(e),e)\downarrow$. Let $B = B_{h(e)}$. It is not hard to check that h is as above.

Having concluded the construction, we first verify that B is promptly simple. Clearly, B is c.e., and it is coinfinite because at most one number is put into B for the sake of each $\mathcal{R}_e$, and that number must be greater than $2e$. As argued above, it is now enough to show that each $\mathcal{R}_e$ is met. So suppose that $|W_e| = \infty$. Let u be such that $f_t(h(e)) = f(h(e))$ for all $t \geqslant u$. There must be some $x > \max\{2e, h(e), u\}$ and some s such that $x \in W_e[s] \setminus W_e[s-1]$. If $\mathcal{R}_e$ is not yet met at stage s, then x will enter B at stage s, thus meeting $\mathcal{R}_e$.

We now show that $B \leqslant_{\mathrm{T}} f$. Let

$$q(x) = \mu u > x\, \forall y \leqslant x\, (f_u(y) = f(y)).$$

Then $q \leqslant_{\mathrm{T}} f$. We claim that $x \in B$ iff $x \in B_{q(x)}$. Suppose for a contradiction that $x \in B \setminus B_{q(x)}$. Then x must have entered B for the sake of some $\mathcal{R}_e$ at some stage $s > q(x)$. Thus $f_s(h(e)) = f_t(h(e))$ for all t with $x \leqslant t \leqslant s$. However, $x < q(x) < s$, so $W_{h(e)} = W_{f_s(h(e))} = W_{f_{q(x)}(h(e))} = W_{f(h(e))}$, contradicting the fact that f is fixed-point free. □

As Kučera [216] noted, Theorem 2.22.5 can be used to give a priority-free solution to Post's Problem: Let A be a low PA degree (which exists by the low basis theorem). By Theorem 2.22.2, A computes a DNC function, so by Theorem 2.22.1, A computes a fixed-point free function. Thus A computes a promptly simple c.e. set, which is noncomputable and low, and hence a solution to Post's Problem.

Prompt simplicity also has some striking structural consequences. A c.e. degree $\mathbf{a}$ is *cappable* if there is a c.e. degree $\mathbf{b} > \mathbf{0}$ such that $\mathbf{a} \cap \mathbf{b} = \mathbf{0}$. Otherwise, $\mathbf{a}$ is *noncappable.*

Theorem 2.22.6 (Ambos-Spies, Jockusch, Shore, and Soare [7]). *Every promptly simple degree is noncappable.*

Proof. Let A be promptly simple, with witness function f. Let B be a noncomputable c.e. set. We must build $C \leqslant_{\mathrm{T}} A, B$ meeting

$$\mathcal{R}_e : W_e \neq \overline{C}$$

for all e. To do so we build auxiliary c.e. sets $V_e = W_{h(e)}$ with indices $h(e)$ given by the recursion theorem with parameters. Analyzing the proof of the recursion theorem, we see that there is a computable function g such that for almost all n, if we decide to put n into V_e at a stage s, then n enters $W_{h(e)}$ before stage $g(s)$.

Our strategy for meeting $\mathcal{R}_e$ is straightforward. We pick a follower n targeted for C. If we ever see a stage s such that $n \in W_{e,s}$, we declare n to be *active*. If $W_e \cap C_s = \emptyset$, then we pick a new follower $n' > n$. For an active n, if some number less than n enters B at a stage t, we enumerate n into V_e. If $n \in A_{f(g(t))}$ then we put n into C. In any case, we declare n to be inactive.

Assume for a contradiction that $W_e = \overline{C}$. Then infinitely many followers become active during the construction. Since B is noncomputable, infinitely many of these enter $W_{h(e)}$, so by the prompt simplicity of A, some active follower enters C, whence $W_e \neq \overline{C}$.

The strategies for different requirements act independently, choosing followers in such a way that no two strategies ever have the same follower. Thus all requirements are met. Whenever we enumerate n into C at a stage t, this action is permitted by a change in B below n at stage t and by n itself entering A by stage $f(g(t))$. Since f is computable, $C \leqslant_{\mathrm{T}} A, B$. □

Ambos-Spies, Jockusch, Shore, and Soare [7] extended the above result by showing that the promptly simple degrees and the cappable degrees form an algebraic decomposition of the c.e. degrees into a strong filter and an ideal. (See [7] for definitions.) Furthermore, they showed that the promptly simple degrees coincide with the low cuppable degrees, where $\mathbf{a}$ is *low cuppable* if there is a low c.e. degree $\mathbf{b}$ such that $\mathbf{a} \cup \mathbf{b} = \mathbf{0}'$. The proofs of these results are technical, and would take us too far afield.

Corollary 2.22.7 (Kučera [216]). *Let* $\mathbf{a}$ *and* $\mathbf{b}$ *be* Δ^0_2 *degrees both of which compute fixed-point free functions. Then* $\mathbf{a}$ *and* $\mathbf{b}$ *do not form a minimal pair.*

Proof. By Theorem 2.22.5, each of $\mathbf{a}$ and $\mathbf{b}$ bounds a promptly simple degree. By Theorem 2.22.6, these promptly simple degrees are noncappable, and hence do not form a minimal pair. Thus neither do $\mathbf{a}$ and $\mathbf{b}$.[20] □

Kučera [217] introduced the following variation of the notion of DNC function.

Definition 2.22.8 (Kučera [217]). A total function f is *generally noncomputable* (*GNC*) if $f(\langle e, n\rangle) \neq \Phi_e(n)$ for all e and n.

We can argue as before to show that the degrees computing $\{0,1\}$-valued GNC functions are exactly the degrees computing $\{0,1\}$-valued DNC functions, that is, the PA degrees.

[20] Kučera's original proof of this result was direct, in the style of the proof of Theorem 2.22.5, and used the (double) recursion theorem.

Theorem 2.22.9 (Kučera [217]). *Let A be a Π^0_1 class whose elements are all $\{0,1\}$-valued GNC functions, and let C be a set. There is a function $g \in A$ and an index e such that $g(\langle e,n\rangle) = C(n)$ for all n. Furthermore, e can be found effectively from an index for A.*

Proof. Using the recursion theorem, choose e such that $\Phi_e(n)$ is defined as follows. Suppose there is a $\tau \in 2^{<\omega}$ such that

$$A \cap \{f : \forall n < |\tau|\,(f(\langle e,n\rangle) = \tau(n))\} = \emptyset.$$

This condition is c.e., so we can choose the first τ we see satisfying it and let $\Phi_e(n) = 1 - \tau(n)$ for $n < |\tau|$ and $\Phi_e(n) = 0$ for $n \geqslant |\tau|$. If there is no such τ, then let $\Phi_e(n)\uparrow$ for all n.

If there is such a τ, then for each $\{0,1\}$-valued GNC function f we have $f(\langle e,n\rangle) = \tau(n)$ for all $n < |\tau|$, and hence $A \cap \{f : \forall n < |\tau|\,(f(\langle e,n\rangle) = \tau(n))\} = A \neq \emptyset$, contradicting the definition of τ. Thus there is no such τ, and hence for every σ there is a $g \in A$ such that $g(\langle e,n\rangle) = \sigma(n)$ for all $n < |\sigma|$. By compactness, there is a $g \in A$ such that $g(\langle e,n\rangle) = C(n)$ for all n. □

Kučera [217] remarked that the use of $\langle e,n\rangle$ with $\Phi_e(n)\uparrow$ for all n is an analog of the use of what are known as flexible formulas in axiomatizable theories, first introduced and studied by Mostowski [283] and Kripke [214], in connection with Gödel's Incompleteness Theorem.

In unpublished work,[21] Kučera also noted that the above proof can be combined with the proof of the low basis theorem to show that if $\mathbf{c}$ is a low degree then there is a low PA degree $\mathbf{a} \geqslant \mathbf{c}$. Combining this result with Kučera's priority-free solution to Post's Problem, we see that for any low degree $\mathbf{c}$ there is a c.e. degree $\mathbf{b}$ such that $\mathbf{b} \cup \mathbf{c}$ is low. This result is particularly interesting in light of the fact that Lewis [245] has constructed a minimal (and hence low$_2$) degree $\mathbf{c}$ such that $\mathbf{b} \cup \mathbf{c} = \mathbf{0}'$ for every nonzero c.e. degree $\mathbf{b}$. Kučera has also used the above method to construct PA degrees in various jump classes.

2.23 Array noncomputability and traceability

In this section, we discuss the *array noncomputable degrees* introduced by Downey, Jockusch, and Stob [120, 121]. The original definition of this class was in terms of *very strong arrays*. Recall that a strong array is a computable collection of pairwise disjoint finite sets $\{F_n : n \in \mathbb{N}\}$ (which recall means not only that the F_n are uniformly computable, but also that the function $i \mapsto \max F_i$ is computable).

[21] This work is now mentioned as example 2.1 in Kučera and Slaman [222]. In unpublished work, Kučera and Slaman have also shown that there is a low$_2$ degree that joins all Δ^0_2 fixed-point free degrees to $\mathbf{0}'$.

Definition 2.23.1 (Downey, Jockusch, and Stob [120]).

(i) A strong array $\{F_n : n \in \mathbb{N}\}$ is a *very strong array* if $|F_n| > |F_m|$ for all $n > m$.

(ii) For a very strong array $\mathcal{F} = \{F_n : n \in \mathbb{N}\}$, we say that a c.e. set A is *$\mathcal{F}$-array noncomputable* (*$\mathcal{F}$-a.n.c.*) if for each c.e. set W there is a k such that $W \cap F_k = A \cap F_k$.

Notice that for any c.e. set W and any k there is a c.e. set $\widehat{W}$ such that $W \cap F_k \neq \widehat{W} \cap F_k$ but $W(n) = \widehat{W}(n)$ for $n \notin F_k$. From this observation it follows that if A is $\mathcal{F}$-a.n.c. then for each c.e. set W there are infinitely many k such that $W \cap F_k = A \cap F_k$.

This definition was designed to capture a certain kind of multiple permitting construction. The intuition is that for A to be $\mathcal{F}$-a.n.c., A needs $|F_k|$ many permissions to agree with W on F_k. As we will see below, up to degree, the choice of very strong array does not matter. Downey, Jockusch, and Stob [120] used multiple permitting to show that several properties are equivalent to array noncomputability. For example, the a.n.c. c.e. degrees are precisely those that bound c.e. sets A_1, A_2, B_1, B_2 such that $A_1 \cap A_2 = B_1 \cap B_2 = \emptyset$ and every separating set for A_1, A_2 is Turing incomparable with every separating set for B_1, B_2. (There are a number of other characterizations of the array noncomputable c.e. degrees in [120, 121].) We will see two examples of this technique connected with algorithmic randomness in Theorems 9.14.4 and 9.14.7 below.

In Downey, Jockusch, and Stob [121], a new definition of array noncomputability was introduced, based on domination properties of functions and not restricted to c.e. sets.

Definition 2.23.2 (Downey, Jockusch, and Stob [121]). A degree $\mathbf{a}$ is *array noncomputable* (*a.n.c*) if for each $f \leqslant_{\mathrm{wtt}} \emptyset'$ there is a function g computable in $\mathbf{a}$ such that $g(n) \geqslant f(n)$ for infinitely many n. Otherwise, $\mathbf{a}$ is *array computable*.

Note that, using this definition, the a.n.c. degrees are clearly closed upwards. The following results give further characterizations of array noncomputability. They also show that, for c.e. sets, the two definitions of array noncomputability coincide and the first definition is independent of the choice of very strong array up to degree. Let $m_{\emptyset'}(n)$ be the least s such that $\emptyset' \upharpoonright n = \emptyset'_s \upharpoonright n$.

Theorem 2.23.3 (Downey, Jockusch, and Stob [121]). *Let* $\mathbf{a}$ *be a degree, and let* $\{F_n\}_{n\in\omega}$ *be a very strong array. The following are equivalent.*

(i) *The degree* $\mathbf{a}$ *is a.n.c.*

(ii) *There is a function h computable in* $\mathbf{a}$ *such that $h(n) \geqslant m_{\emptyset'}(n)$ for infinitely many n.*

(iii) *There is a function g computable in* $\mathbf{a}$ *such that for each e there is an n for which $W_e \cap F_n = W_{e,g(n)} \cap F_n$.*

Proof. To prove that (i) → (iii), let $f(n) = \mu s\, \forall e \leqslant n\, (W_e \cap F_n = W_{e,s} \cap F_n)$. Then $f \leqslant_{\mathrm{wtt}} \emptyset'$, so there is an $\mathbf{a}$-computable function g such that $g(n) \geqslant f(n)$ for infinitely many n.

To prove that (ii) → (i), let $f \leqslant_{\mathrm{wtt}} \emptyset'$, and let h satisfy (ii). Fix e and a computable function b such that $f(n) = \Phi_e^{\emptyset'}(n)$ with use at most $b(n)$ for all n. We may assume without loss of generality that h and b are increasing. We define a function g such that $g(n) \geqslant f(n)$ for infinitely many n as follows. Given n, let s be minimal such that $s > h(b(n+1))$ and $\Phi_e^{\emptyset'}(n)[s]\downarrow$ with use at most $b(n)$, and let $g(n) = \Phi_e^{\emptyset'}(n)[s]$. Clearly g is computable in $\mathbf{a}$. Let n and k be such that $b(n) \leqslant k \leqslant b(n+1)$ and $h(k) \geqslant m_{\emptyset'}(k)$, and let s be as in the definition of $g(n)$. We have $s \geqslant h(b(n+1)) \geqslant h(k) \geqslant m_{\emptyset'}(k) \geqslant m_{\emptyset'}(b(n))$, so $\emptyset'_s \upharpoonright \varphi_e^{\emptyset'}(n)[s] = \emptyset' \upharpoonright \varphi_e^{\emptyset'}(n)[s]$, and hence $g(n) = f(n)$. Since there are infinitely many j with $h(j) \geqslant m_{\emptyset'}(j)$, there are infinitely many n for which there is a k as above, and hence $g(n) = f(n)$ for infinitely many n.

It remains to show that (iii) → (ii). Let g be as in (iii). We claim that for each e there are infinitely many n with $W_e \cap F_n = W_{e,g(n)} \cap F_n$. Suppose otherwise. Let i be such that Φ_i is defined as follows. If $x \in F_n$ for one of the finitely many n such that $W_e \cap F_n = W_{e,g(n)} \cap F_n$, then let $\Phi_i(x)$ converge in strictly more than $g(n)$ steps. Otherwise, if $\Phi_e(x)\downarrow$ then let $\Phi_i(x)$ converge in at least the same number of steps as $\Phi_e(x)$, and if $\Phi_e(x)\uparrow$ then let $\Phi_i(x)\uparrow$. Then there is no n such that $W_i \cap F_n = W_{i,g(n)} \cap F_n$, contradicting the choice of g. Thus the claim holds, so it suffices to show that there is an e such that $\mu s\, (W_{e,s} \cap F_{n+1} = W_e \cap F_{n+1}) \geqslant m_{\emptyset'}(n)$ for all n, since it then follows that (ii) holds with $h(n) = g(n+1)$.

Define a c.e. set V as follows. At stage s, let $c_{n,s}$ be the least element (if any) of $F_{n+1} \setminus V_s$. For each $n < s$, if $\emptyset'_{s+1} \upharpoonright n \neq \emptyset'_s \upharpoonright n$ then enumerate $c_{n,s}$ into V. Also enumerate $c_{s,s}$ into V.

Note that $|F_{n+1} \cap V| \leqslant |\{s : \exists i < n\, (i \in \emptyset'_{s+1} \setminus \emptyset'_s)\}| + 1 \leqslant n+1 < |F_{n+1}|$, so $c_{n,s}$ is defined for all n and s. It follows that $\mu s\, (V_s \cap F_{n+1} = V \cap F_{n+1}) \geqslant m_{\emptyset'}(n)$. By Theorem 2.3.3, there is an e such that $W_e = V$ and $W_{e,s} \subseteq V_s$ for all s. Then $\mu s\, (W_{e,s} \cap F_{n+1} = W_e \cap F_{n+1}) \geqslant \mu s\, (V_s \cap F_{n+1} = V \cap F_{n+1}) \geqslant m_{\emptyset'}(n)$ for all n, as required. □

Since the truth of item (i) in Theorem 2.23.3 does not depend on a choice of very strong array, neither does the truth of item (iii).

Theorem 2.23.4 (Downey, Jockusch, and Stob [120, 121]). *Let* $\mathbf{a}$ *be a c.e. degree and let $\mathcal{F} = \{F_n\}_{n\in\mathbb{N}}$ be a very strong array. Then* $\mathbf{a}$ *is a.n.c. in the sense of Definition 2.23.2 iff there is an $\mathcal{F}$-a.n.c. c.e. set $A \in$* $\mathbf{a}$.

Proof. (⇐) Let $A \in \mathbf{a}$ be an $\mathcal{F}$-a.n.c. c.e. set. We show that item (iii) in Theorem 2.23.3 holds with $g(n) = \mu s\, (A_s \cap F_n = A \cap F_n)$. For each e, let $V_e = \{x : \exists s\, (x \in W_{e,s} \setminus A_s)\}$. Each V_e is c.e., so there is an n such that $A \cap F_n = V_e \cap F_n$. If x enters $W_e \cap F_n$ after stage $g(n)$, then we have two cases: if $x \in A$ then $x \in A_{g(n)}$, so $x \notin V_e$, while if $x \notin A$ then $x \in V_e$. Both these possibilities contradict the fact that $A \cap F_n = V_e \cap F_n$, so $W_e \cap F_n = W_{e,g(n)} \cap F_n$.

($\Rightarrow$) Let $\mathbf{a}$ be an a.n.c. c.e. degree and let $\{F_n\}_{n\in\mathbb{N}}$ be a very strong array. Let $f(n) = \mu s\, \forall e \leqslant n\, (W_{e,s} \cap F_{\langle e+1,n\rangle} = W_e \cap F_{\langle e+1,n\rangle})$. Clearly $f \leqslant_{\mathrm{wtt}} \emptyset'$, so there is an $\mathbf{a}$-computable g such that $g(n) \geqslant f(n)$ for infinitely many n. Since g is $\mathbf{a}$-computable and $\mathbf{a}$ is c.e., there is a computable function $h(n,s)$ and an $\mathbf{a}$-computable function p such that $g(n) = h(n,s)$ for all $s \geqslant p(n)$. We now define the c.e. set A.

First, we let $B \in \mathbf{a}$ be c.e. and put $\bigcup_{n\in B} F_{\langle 0,n\rangle}$ into A. We will not put any other elements of any $F_{\langle 0,n\rangle}$ into A, so this action ensures that A has degree at least $\mathbf{a}$.

Now it suffices to ensure that if $n \geqslant e$ and $g(n) \geqslant f(n)$, then $A \cap F_{\langle e+1,n\rangle} = W_e \cap F_{\langle e+1,n\rangle}$. Whenever $h(n,s) \neq h(n,s+1)$ and $e \leqslant n \leqslant s$, we put all elements of $W_{e,h(n,s+1)} \cap F_{\langle e+1,n\rangle}$ into A at stage s.

The definition of A ensures that if $x \in A \cap F_{\langle e+1,n\rangle}$ then $x \in A_{p(n)}$, so A is computable in p and hence $A \in \mathbf{a}$. Suppose now that $n \geqslant e$ and $g(n) \geqslant f(n)$. It follows from the definition of f that $W_{e,g(n)} \cap F_{\langle e+1,n\rangle} = W_e \cap F_{\langle e+1,n\rangle}$. Choose s as large as possible so that $h(n,s) \neq h(n,s+1)$. (There is no loss of generality in assuming there is at least one such $s \geqslant n$.) Then $h(n,s+1) = g(n)$ and so

$$\begin{aligned} A_{s+1} \cap F_{\langle e+1,n\rangle} &= W_{e,h(n,s+1)} \cap F_{\langle e+1,n\rangle} \\ &= W_{e+1,g(n)} \cap F_{\langle e+1,n\rangle} = W_e \cap F_{\langle e+1,n\rangle}. \end{aligned}$$

Furthermore, by the maximality of s, no elements of $F_{\langle e+1,n\rangle}$ enter A after stage $s+1$, so $A \cap F_{\langle e+1,n\rangle} = W_e \cap F_{\langle e+1,n\rangle}$. ☐

Corollary 2.23.5 (Downey, Jockusch, and Stob [120, 121]). *Let $\mathcal{F}$ and $\mathcal{G}$ be very strong arrays. Then a degree contains an $\mathcal{F}$-a.n.c. c.e. set iff it contains a $\mathcal{G}$-a.n.c. c.e. set.*

We do not need to consider all wtt-reductions in the definition of array noncomputability, but can restrict ourselves to reductions with use bounded by the identity function.

Proposition 2.23.6 (Downey and Hirschfeldt, generalizing [120]). *A degree $\mathbf{a}$ is a.n.c. iff for every f that is computable from $\emptyset'$ via a reduction with use bounded by the identity function, there is an $\mathbf{a}$-computable function g such that $g(n) \geqslant f(n)$ for infinitely many n.*

Proof. The "only if" direction is obvious. We prove the "if" direction. Suppose that $\Gamma^{\emptyset'} = f$ with use bounded by the increasing computable function p. Let $\Phi^{\emptyset'}(n) = \max\{\Gamma^{\emptyset'}(m) : p(m) \leqslant n\}$. Then $\varphi^{\emptyset'}(n) \leqslant n$, so there is an $\mathbf{a}$-computable increasing function g such that $g(n) \geqslant \Phi^{\emptyset'}(n)$ for infinitely many n. Let $\widehat{g}(m) = g(p(m+1))$. Since p is computable, $\widehat{g} \leqslant_{\mathrm{T}} A$. Furthermore, if $g(n) \geqslant \Phi^{\emptyset'}(n)$ then for the largest m such that $p(m) \leqslant n$, we have $\widehat{g}(m) = g(p(m+1)) > g(n) \geqslant \Phi^{\emptyset'}(n) \geqslant \Gamma^{\emptyset'}(m) = f(m)$, so $\widehat{g}(m) \geqslant f(m)$ for infinitely many m. □

Martin [258] gave a characterization of the GL_2 sets quite close to the definition of array computability, as a consequence of the following result. Recall that a function f dominates a function g if $f(n) \geqslant g(n)$ for all sufficiently large n. A function is *dominant* if it dominates all computable functions.

Theorem 2.23.7 (Martin [258]). *A set A is high iff there is a dominant function $f \leqslant_{\mathrm{T}} A$.*

Proof. Recall that $\mathrm{Tot} = \{e : \Phi_e \text{ is total}\}$. This set has degree $\emptyset''$, so A is high iff $\mathrm{Tot} \leqslant_{\mathrm{T}} A'$ iff there is an A-computable function h such that $\lim_s h(e,s) = \mathrm{Tot}(e)$ for all e.

Suppose that A is high, and let h be as above. Define an A-computable f as follows. Given n, for each $e \leqslant n$, look for an $s \geqslant n$ such that either

1. $\Phi_e(n)[s]\downarrow$ or
2. $h(e,s) = 0$.

For each $e \leqslant n$, one of the two must happen. Define $f(n)$ to be larger than $\Phi_e(n)$ for all $e \leqslant n$ such that 1 holds. If Φ_e is total then $h(e,s) = 1$ for all sufficiently large s, so $f(n) > \Phi_e(n)$ for all sufficiently large n. Thus f dominates all computable functions.

Now suppose that there is a function $f \leqslant_{\mathrm{T}} A$ that dominates all computable functions. Define $h \leqslant_{\mathrm{T}} A$ as follows. Given e and s, check whether $\Phi_e(n)[f(s)]\downarrow$ for all $n \leqslant s$. If so, let $h(e,s) = 1$, and otherwise let $h(e,s) = 0$. If $e \notin \mathrm{Tot}$ then clearly $h(e,s) = 0$ for all sufficiently large s. If $e \in \mathrm{Tot}$ and s is sufficiently large then $f(s)$ is greater than the stage by which $\Phi_e(n)$ converges for all $n \leqslant s$, since the latter is a computable function of s. So $h(e,s) = 1$ for all sufficiently large s. Thus $\lim_s h(e,s) = \mathrm{Tot}(e)$ for all e. □

Corollary 2.23.8 (Martin [258]). *A set A is GL_2 iff there is a function $f \leqslant_{\mathrm{T}} A \oplus \emptyset'$ that dominates all A-computable functions.*

Proof. A set A is GL_2 iff $(A \oplus \emptyset')' \equiv_{\mathrm{T}} A''$ iff $A \oplus \emptyset'$ is high relative to A. Thus the corollary follows by relativizing Theorem 2.23.7 to A. □

By Corollary 2.23.8, if a set A is not GL_2 then for each function $f \leqslant_{\mathrm{T}} A \oplus \emptyset'$ there is a function $g \leqslant_{\mathrm{T}} A$ such that $g(n) \geqslant f(n)$ for infinitely many n. Thus if a degree is not GL_2 then it is a.n.c. In particular, every array computable Δ^0_2 degree is low_2. In [121], Downey, Jockusch, and Stob demonstrated that many results previously proved for non-GL_2 degrees extend to a.n.c. degrees. However, the a.n.c degrees do not coincide with the non-GL$_2$ degrees, nor do the a.n.c. c.e. degrees coincide with the nonlow$_2$ c.e. degrees.

Theorem 2.23.9 (Downey, Jockusch, and Stob [120]). *There is a low a.n.c. c.e. degree.*

Proof. The proof is a straightforward combination of lowness and array noncomputability requirements. Let $\mathcal{F} = \{F_n : n \in \mathbb{N}\}$ be a very strong array with $|F_n| = n + 1$. We build a c.e. set A to satisfy the requirements

$$\mathcal{R}_e : \exists n\,(W_e \cap F_n = A \cap F_n)$$

and

$$\mathcal{N}_e : \exists^\infty s\, \Phi_e^A(e)[s]\downarrow \Rightarrow\ \Phi_e^A(e)\downarrow .$$

For the sake of $\mathcal{R}_e$ we simply pick a fresh n to devote to making $W_e \cap F_n = A \cap F_n$. We do so in the obvious way: Whenever a new element enters $W_e \cap F_n$, we put it into A. Such enumerations will happen at most $n + 1$ many times. Each time one happens we initialize the weaker priority $\mathcal{N}$-requirements. Similarly, each time we see a computation $\Phi_e^A(e)[s]\downarrow$ we initialize the weaker priority $\mathcal{R}$-requirements, which now must pick their n's to be above the use of this computation. The result follows by the usual finite injury argument. □

On the other hand, an easy modification of Martin's proofs above shows that if A is superlow then there is a function wtt-computable in $\emptyset'$ that dominates every A-computable function, and hence A is array computable.

Array computability is also related to hyperimmunity.

Proposition 2.23.10 (Folklore). *If* **a** *is hyperimmune-free then* **a** *is array computable.*

Proof. If A is of hyperimmune-free degree then every $f \leqslant_{\mathrm{T}} A$ is dominated by some total computable function. It is easy to construct a function $g \leqslant_{\mathrm{wtt}} \emptyset'$ that is a uniform bound for all partial computable functions. (That is, for each e, we have $\Phi_e(n) \leqslant h(n)$ for almost all n such that $\Phi_e(n)\downarrow$.) Then every function $f \leqslant_{\mathrm{T}} A$ is dominated by g, and hence A is array computable. □

The following concept, closely related to both hyperimmune-freeness and array computability, is quite useful in the study of algorithmic randomness.

Definition 2.23.11 (Zambella [416], see [386], also Ishmukhametov [184]). A set A, and the degree of A, are *c.e. traceable* if there is a computable function p (called a *bound*) such that, for each function $f \leqslant_{\mathrm{T}} A$, there is a computable function h (called a *trace* for f) satisfying, for all n,

(i) $|W_{h(n)}| \leqslant p(n)$ and

(ii) $f(n) \in W_{h(n)}$.

Since one can uniformly enumerate all c.e. traces for a fixed bound p, there is a universal trace with bound p^2, say. By a *universal trace* we mean one that traces each function $f \leqslant_{\mathrm{T}} A$ on almost all inputs. As the following result shows, the above definition does not change if we replace "there is a computable function p" by "for every unbounded nondecreasing computable function p such that $p(0) > 0$".

Proposition 2.23.12 (Terwijn and Zambella [389]). *Let A be c.e. traceable and let q be an unbounded nondecreasing computable function such that $q(0) > 0$. Then A is c.e. traceable with bound q.*

Proof. Let A be c.e. traceable with some bound p. Let $g \leqslant_{\mathrm{T}} A$. For each m, let r_m be such that $q(r_m) \geqslant p(m+1)$, and let $f(m) = \langle g(0), \ldots, g(r_m)\rangle$. Let h be as in Definition 2.23.11. Let $\widehat{h}$ be a computable function such that $W_{\widehat{h}(n)}$ is defined as follows. For each m, if $r_m \leqslant n < r_{m+1}$, then enumerate into $W_{\widehat{h}(n)}$ all the nth coordinates of elements of $W_{h(m+1)}$. Then $|W_{\widehat{h}(n)}| \leqslant |W_{h(m+1)}| \leqslant p(m+1) \leqslant q(r_m) \leqslant q(n)$. Furthermore, $g(n)$ is the nth coordinate of $f(m+1)$, and $f(m+1) \in W_{h(m+1)}$, so $g(n) \in W_{\widehat{h}(n)}$. We have not yet defined $W_{\widehat{h}(n)}$ for $n < r_0$, but since there are only finitely many such n, we can define $W_{\widehat{h}(n)} = \{g(n)\}$ for such n. Then for all n we have $|W_{\widehat{h}(n)}| \leqslant q(n)$ and $g(n) \in W_{\widehat{h}(n)}$, as required. □

Ishmukhametov [184] gave the following characterization of the array computable c.e. degrees.

Theorem 2.23.13 (Ishmukhametov [184]). *A c.e. degree is array computable iff it is c.e. traceable.*

Proof. Let A be a c.e. traceable c.e. set. Let $f \leqslant_{\mathrm{T}} A$. Let p and h be as in Definition 2.23.11. Let $g(n) = \max W_{h(n)}$. Then $g \leqslant_{\mathrm{wtt}} \emptyset'$, since we can use $p(n)$ to bound the $\emptyset'$-oracle queries needed to compute $g(n)$, and g dominates f. Since f is arbitrary, A is array computable.

Now let A be an array computable c.e. set. By Proposition 2.23.6, there is a function g computable from $\emptyset'$ with use bounded by the identity that dominates every A-computable function. Let $f = \Phi_e^A$. Let $F(n)$ be the least s such that $\Phi_e^A(n)[t] = \Phi_e^A(n)[s]$ for all $t > s$. Then $F \leqslant_{\mathrm{T}} A$, so by modifying g at finitely many places, we obtain a function G that is computable in $\emptyset'$ with use bounded by the identity function and such that $G(n) > F(n)$ for all n. Since $\emptyset' \upharpoonright n$ can change at most n many times during the enumeration of $\emptyset'$, there is a computable approximation $G_0, G_1, \ldots$ to G that changes value at n at most n times. Let h be a computable function such that $W_{h(n)} = \{\Phi_e^A(n)[G_s(n)] : s \in \mathbb{N} \wedge \Phi_e^A(n)[G_s(n)]\downarrow\}$. Then $|W_{h(n)}| \leqslant n$ and $f(n) \in W_{h(n)}$. Since f is arbitrary, A is c.e. traceable. □

Using this characterization, Ishmukhametov proved the following remarkable theorem, which shows that if the c.e. degrees are definable in the global structure of the degrees, then so are the a.n.c. c.e. degrees. A degree $\mathbf{m}$ is a *strong minimal cover* of a degree $\mathbf{a} < \mathbf{m}$ if for all degrees $\mathbf{d} < \mathbf{m}$, we have $\mathbf{d} \leqslant \mathbf{a}$.

Theorem 2.23.14 (Ishmukhametov [184]). *A c.e. degree is array computable iff it has a strong minimal cover.*

Definition 2.23.11 can be strengthened as follows. Recall that $D_0, D_1, \ldots$ is a canonical listing of finite sets, as defined in Chapter 1.

Definition 2.23.15 (Terwijn and Zambella [389]). A set A, and the degree of A, are *computably traceable* if there is a computable function p such that, for each function $f \leqslant_{\mathrm{T}} A$, there is a computable function h satisfying, for all n,

(i) $|D_{h(n)}| \leqslant p(n)$ and

(ii) $f(n) \in D_{h(n)}$.

As with c.e. traceability, the above definition does not change if we replace "there is a computable function p" by "for every unbounded nondecreasing computable function p such that $p(0) > 0$".

If A is computably traceable then each function $g \leqslant_{\mathrm{T}} A$ is dominated by the function $f(n) = \max D_{h(n)}$, where h is a trace for f. Thus, every computably traceable degree is hyperimmune-free. One may think of computable traceability as a uniform version of hyperimmune-freeness. Terwijn and Zambella [389] showed that a simple variation of the standard construction of hyperimmune-free sets by Miller and Martin [282] (Theorem 2.17.3) produces continuum many computably traceable sets. Indeed, the proof we gave of Theorem 2.17.3 produces a computably traceable set.

2.24 Genericity and weak genericity

The notion of genericity arose in the context of set-theoretic forcing. It was subsequently "miniaturized" to obtain a notion of n-genericity appropriate to computability theory. Although the original definition of n-genericity was in terms of forcing, the following more modern definition is the one that is now generally used.[22] Let $S \subseteq 2^{<\omega}$. A set A *meets* S if there is an

[22]The original definition of n-genericity, dating back to Feferman [143], is as follows. Let $\mathcal{L}$ be the usual first-order language of number theory together with a set constant X and the membership relation $\in$. Let $\sigma \in 2^{<\omega}$. For sentences ψ in $\mathcal{L}$, we define the notion "σ *forces* φ", written $\sigma \Vdash \psi$, by recursion on the structure of ψ as follows.

(i) If ψ is atomic and does not contain X, then $\sigma \Vdash \psi$ iff ψ is true in arithmetic.

(ii) $\sigma \Vdash n \in X$ iff $\sigma(n) = 1$.

(iii) $\sigma \Vdash \psi_0 \vee \psi_1$ iff $\sigma \Vdash \psi_0$ or $\sigma \Vdash \psi_1$.

(iv) $\sigma \Vdash \neg\psi$ iff $\forall \tau \succcurlyeq \sigma\, (\tau \nVdash \psi)$.

n such that $A \upharpoonright n \in S$. A set A *avoids* S if there is an n such that for all $\tau \succcurlyeq A \upharpoonright n$, we have $\tau \notin S$.

Definition 2.24.1 (Jockusch and Posner, see Jockusch [188])**.** A set is *n-generic* if it meets or avoids each Σ_n^0 set of strings.

A set is *arithmetically generic* if it is n-generic for all n.

It is easy to construct n-generic Δ_{n+1}^0 sets, and arithmetically generic sets computable from $\emptyset^{(\omega)}$. Indeed, it does not require much computational power to build a 1-generic set.

Proposition 2.24.2. *Every noncomputable c.e. set computes a* 1*-generic set.*

Proof. The proof is a standard permitting argument. Let $S_0, S_1, \dots$ be an effective listing of the c.e. sets of strings. We build $G \leqslant_{\mathrm{T}} A$ to satisfy the requirements

$$R_e : G \text{ meets or avoids } S_e.$$

Associated with each R_e is a restraint r_e, initially set to 0. We say that R_e requires attention through n at stage s if $n > r_e$ and $G_s \upharpoonright n$ has an extension in $S_e[s]$. For each e, we also have a set M_e of numbers through which R_e has required attention.

Initially, we have $G_0 = \emptyset$. At stage s, let $e \leqslant s$ be least such that R_e is not currently satisfied and either

1. there is an $n > r_e$ such that $G_s \upharpoonright n$ has an extension σ in $S_e[s]$ and $A_{s+1} \upharpoonright n \neq A_s \upharpoonright n$ or

2. R_e requires attention through some n not currently in M_e.

(If there is no such e then let $G_{s+1} = G_s$ and proceed to the next stage.) In the first case, let G_{s+1} extend σ and declare R_e to be satisfied. In the second, put n into M_e and let $G_{s+1} = G_s$. In either case, for all $i > e$, declare R_i to be unsatisfied, empty M_i, and let r_i be a fresh large number. We say we have acted for e.

Let $G = \lim_s G_s$. Once numbers less than n have stopped being enumerated into A, we cannot change G below n, so G is well-defined and A-computable. Assume by induction that there is a stage s after which we never act for any $i < e$. If we ever act for e after stage s and are in case 1

(v) $\sigma \Vdash \exists z\, \psi(z)$ iff there is an $n \in \mathbb{N}$ such that $\sigma \Vdash \psi(n)$.

For a set A, we have $A \Vdash \psi$ iff there is some $\sigma \prec A$ such that $\sigma \Vdash \psi$.

A set A is *n-generic* if for each Σ_n^0 sentence ψ of $\mathcal{L}$, either $A \Vdash \psi$ or $A \Vdash \neg\psi$. As Jockusch and Posner showed (see Jockusch [188]), it is not hard to see that this definition is equivalent to the one in Definition 2.24.1 using the result due to Matijacevic [261] that each Σ_n^0 subset of $\mathbb{N}$ is defined by a Σ_n^0 formula in $\mathcal{L}$.

of the construction, we permanently satisfy R_e by ensuring that G meets S_e. In that case, R_e never again acts, so the induction can proceed.

So suppose we never act for e in case 1 of the construction after stage s. Assume for a contradiction that we act infinitely often for e. At each stage $t > s$ at which we act for e, we find a new n such that $G_t \upharpoonright n$ has an extension in $S_e[t]$. Since we then increase all restraints for weaker priority requirements, $G \upharpoonright n = G_t \upharpoonright n$. Since we never reach case 1 of the construction for e, we must have $A \upharpoonright n = A_t \upharpoonright n$. From this fact it is easy to see that A is computable, which is a contradiction.

Thus we act only finitely often for e. But if $G \upharpoonright n$ has an extension in S_e, then R_e requires attention through n at all large enough stages t. Thus no sufficiently long initial segment of G has an extension in S_e, so G avoids S_e. ☐

A degree is n-generic if it contains an n-generic set, and arithmetically generic if it contains an arithmetically generic set. An interesting fact about arithmetically generic degrees, shown by Jockusch [188], is that if **a** and **b** are arithmetically generic, then the structures of the degrees below **a** and of the degrees below **b** are elementarily equivalent. There is a wealth of results on n-generic sets and degrees, many of which are discussed in the survey Jockusch [188]. Kumabe [224] also contains several results on n-genericity.

The following is an important property of n-generic sets.

Theorem 2.24.3 (Jockusch [188]). *If A is n-generic then $A^{(n)} \equiv_T A \oplus \emptyset^{(n)}$. In particular, every 1-generic set is GL_1.*

Proof. Assume by induction that the theorem is true for $n-1$. (The base case $n = 0$ is trivial, as it says that $A \equiv_T A \oplus \emptyset$ for all A.) Let A be n-generic. By the inductive hypothesis, it is enough to show that $(A \oplus \emptyset^{(n-1)})' \equiv_T A \oplus \emptyset^{(n)}$. One direction of this equivalence is immediate, so we are left with showing that $(A \oplus \emptyset^{(n-1)})' \leqslant_T A \oplus \emptyset^{(n)}$.

Let $R_e = \{\sigma : \Phi_e^{\sigma \oplus (\emptyset^{(n-1)} \upharpoonright |\sigma|)}(e)\downarrow\}$ and $S_e = \{\sigma : \forall \tau \succcurlyeq \sigma\, (\tau \notin R_e)\}$. The R_e are c.e. in $\emptyset^{(n-1)}$ and hence are Σ_n^0. So for each e, we have $e \in (A \oplus \emptyset^{(n-1)})'$ iff A meets R_e iff A does not meet S_e, the last equivalence following by n-genericity. But it is easy to check that the R_e and S_e are uniformly $\emptyset^{(n)}$-computable, so using $A \oplus \emptyset^{(n)}$, we can compute $(A \oplus \emptyset^{(n-1)})'$. ☐

The above result is an example of the phenomenon that, in many cases, a construction used to show that a set with a certain property (such as being GL_1) exists actually shows that the given property holds of all sufficiently generic sets. For example, the requirements of a finite extension argument are usually ones that ask that the set being constructed meet or avoid some open set of conditions.

A related phenomenon is the fact that genericity requirements can often be added to finite extension constructions. For instance, it is straightforward to modify the construction in the proof of Theorem 2.16.2 to obtain the following result.

Theorem 2.24.4. *If $C > 0$ and $D \geqslant_{\mathrm{T}} \emptyset' \oplus C$ then there is a 1-generic set A such that $A' \equiv_{\mathrm{T}} A \oplus C \equiv_{\mathrm{T}} D$.*

The following result exemplifies the fact that generic sets are quite computationally weak.

Theorem 2.24.5 (Demuth and Kučera [96])**.** *If a set is 1-generic then it does not compute a DNC function.*

Proof. Let G be 1-generic and let $\Phi^G = f$. Let

$$S = \{\sigma \in \omega^{<\omega} : \forall n < |\sigma| \, (\sigma(n) \neq \Phi_n(n))\}.$$

Note that S is co-c.e. Let

$$X = \{\tau : \exists \sigma \, (\Phi^\tau \upharpoonright |\sigma| \downarrow = \sigma \, \wedge \, \sigma \notin S)\}.$$

Then X is c.e., so G either meets or avoids X. If G meets X then there is a $\tau \prec G$ and a $\sigma \in \omega^{<\omega}$ such that $\Phi^\tau \upharpoonright |\sigma| \downarrow = \sigma$ and $\sigma \notin S$. Since $\Phi^G = f$, we have $\sigma \prec f$, and hence f is not DNC. Thus it is enough to assume that G avoids X and derive a contradiction.

Let $\nu \prec G$ be such that if $\tau \in X$ then $\nu \not\prec \tau$. Let

$$Y = \{\sigma : \exists \rho \, (\nu \prec \rho \, \wedge \, \Phi^\rho \upharpoonright |\sigma| \downarrow = \sigma)\}.$$

Then Y is a c.e. subset of S, and it is infinite because every initial segment of f is in Y. So we can computably find $\sigma_0, \sigma_1, \ldots \in S$ such that $|\sigma_i| > i$. Let $g(i) = \sigma_i(i)$. Then g is computable and, by the definition of S, it is DNC, which is a contradiction. □

By Theorem 2.22.2, we have the following result.

Corollary 2.24.6. *No PA degree is computable from a 1-generic degree.*

Since the hyperimmune-free basis theorem implies that there is a DNC function of hyperimmune-free degree, we have the following.

Corollary 2.24.7 (Downey and Yu [137])**.** *There is a hyperimmune-free degree not computable from any 1-generic degree.*

Corollary 2.24.7 should be contrasted with the following result, whose proof uses a special notion of perfect set forcing.

Theorem 2.24.8 (Downey and Yu [137])**.** *There is a hyperimmune-free degree computable from a 1-generic degree.*

Recall that a set A is CEA if it is computably enumerable in some $X <_{\mathrm{T}} A$. By Theorem 2.17.2, no set of hyperimmune-free degree can be CEA. There are also noncomputable Δ^0_2 sets that are not CEA. For example, let A be co-c.e. If A is c.e. in X, then A is computable in X, so A is not CEA.

The following result will be of interest in Section 8.21.3, where we show that being CEA is in fact a property of almost all sets, in the measure-theoretic sense.

Theorem 2.24.9 (Jockusch [188]). *Every* 1*-generic set is CEA.*

Proof.[23] Let A be 1-generic. Let $X = \{2\langle i,j\rangle : i \in A \wedge 2\langle i,j\rangle \notin A\}$. Clearly, $X \leqslant_{\mathrm{T}} A$. For each i, the set $\{\sigma : \exists j\,(\sigma(2\langle i,j\rangle) = 0)\}$ is computable and dense, so for each i there is a j such that $2\langle i,j\rangle \notin A$. Thus $i \in A$ iff $\exists j\,(2\langle i,j\rangle \in X)$, whence A is c.e. in X. So A is CEA(X), and thus we are left with showing that $A \nleqslant_{\mathrm{T}} X$.

Let $\widehat{\sigma}$ be the unique string of length $|\sigma|$ such that $\widehat{\sigma}(n) = 1$ iff $n = 2\langle i,j\rangle$ for i and j such that $\sigma(i) = 1$ and $\sigma(2\langle i,j\rangle) = 0$. We claim that if $\nu \preccurlyeq \sigma$ and $n \geqslant |\nu|$ is odd, then there is a string $\tau \succcurlyeq \nu$ such that $\tau(n) = 1$ and $\widehat{\sigma} \preccurlyeq \widehat{\tau}$.

If $\sigma(n) = 1$ then we can take $\tau = \sigma$, and if $n \geqslant |\sigma|$ then we can take any $\tau \succ \sigma$ such that $\tau(n) = 1$. So suppose that $\sigma(n) = 0$. Let S be the smallest set under inclusion such that $n \in S$, and if $i \in S$, then $2\langle i,j\rangle \in S$ if $\sigma(i) = 0$ and $2\langle i,j\rangle < |\sigma|$. Let τ be the string of length $|\sigma|$ such that $\tau(k) = 1$ iff either $\sigma(k) = 1$ or $k \in S$. Then $\nu \preccurlyeq \tau$. We need to show that $\widehat{\sigma} \preccurlyeq \widehat{\tau}$. Let $k \leqslant |\sigma|$. If k is odd then $\widehat{\sigma}(k) = \widehat{\tau}(k) = 0$, so assume that $k = 2\langle i,j\rangle$ for some i and j.

If $\widehat{\sigma}(k) = 0$, then either $\sigma(i) = 0$ or $\sigma(2\langle i,j\rangle) = 1$. In the latter case, $\tau(2\langle i,j\rangle) = 1$, so $\widehat{\tau}(2\langle i,j\rangle) = 0$. Now suppose that $\sigma(i) = 0$. If $i \in S$, then $2\langle i,j\rangle \in S$, so $\tau(2\langle i,j\rangle) = 1$, and hence $\widehat{\tau}(2\langle i,j\rangle) = 0$. If $i \notin S$, then $\tau(i) = 0$ and hence $\widehat{\tau}(2\langle i,j\rangle) = 0$.

If $\widehat{\sigma}(k) = 1$, then $\sigma(i) = 1$ and $\sigma(2\langle i,j\rangle) = 0$. Since $\sigma(i) = 1$, we have $\tau(i) = 1$. Furthermore, it follows easily by induction that S contains no numbers of the form $2\langle d,q\rangle$ with $\sigma(d) = 1$. Since $\sigma(2\langle i,j\rangle) = 0$ and $2\langle i,j\rangle \notin S$, we have $\tau(2\langle i,j\rangle) = 0$, and hence $\widehat{\tau}(k) = 1$.

We have established our claim. Now assume for a contradiction that $\Phi^X = A$ for some Turing functional Φ. Let $V = \{\sigma : \sigma \mid \Phi^{\widehat{\sigma}}\}$. Then V is c.e. and A extends no string in V, so, since A is 1-generic, there is a $\nu \prec A$ such that no extension of ν is in V. Let $n \geqslant |\nu|$ and $\sigma \succcurlyeq \nu$ be such that n is odd and $\Phi^{\widehat{\sigma}}(n) = 0$. (Such a σ must exist because $\{\rho : \exists n \geqslant |\nu|\,(n \text{ odd} \wedge \rho(n) = 0)\}$ is computable and dense, so there is an odd $n \geqslant |\nu|$ such that $\Phi^X(n) = A(n) = 0$.) Then there is a $\tau \succcurlyeq \nu$ such that $\tau(n) = 1$ and $\widehat{\sigma} \preccurlyeq \widehat{\tau}$. We have $\Phi^{\widehat{\tau}}(n) = 0$ and $\tau(n) = 1$, so $\tau \in V$, contradicting the choice of ν. □

We now consider a weaker notion of genericity. A set of strings D is *dense* if every string has an extension in D. Recall that a set A meets D if there is a $\sigma \in D$ such that $\sigma \prec A$. Similarly, a string τ meets D if there is a

[23]Jockusch [188] attributes the idea behind this proof to Martin.

$\sigma \in D$ such that $\sigma \preccurlyeq \tau$. Given the set-theoretic definition of genericity in terms of meeting dense sets, the following definition is natural.

Definition 2.24.10 (Kurtz [228, 229]). A set is *weakly n-generic* if it meets all dense Σ^0_n sets of strings. A degree is weakly n-generic if it contains a weakly n-generic set.

As noted by Jockusch (see [228, 229]), it is not hard to show that A is weakly $(n+1)$-generic iff A meets all dense Δ^0_{n+1} sets of strings iff A meets all dense Π^0_n sets of strings.

Clearly, if a set is n-generic then it is weakly n-generic. The following result is slightly less obvious.

Theorem 2.24.11 (Kurtz [228, 229]). *Every weakly $(n+1)$-generic set is n-generic.*

Proof. Let A be weakly $(n+1)$-generic. Let S be a Σ^0_n set of strings. Let $R = \{\sigma : \sigma \in S \lor \forall \tau \succcurlyeq \sigma\, (\tau \notin S)\}$. Then R is a dense Σ^0_{n+1} set of strings, and hence A meets R, which clearly implies that A meets or avoids S. Thus A is n-generic. □

Thus we have the following hierarchy: weakly 1-generic $\Leftarrow$ 1-generic $\Leftarrow$ weakly 2-generic $\Leftarrow$ 2-generic $\Leftarrow$ weakly 3-generic $\Leftarrow \cdots$. We now show that none of these implications can be reversed, even for degrees, by exploring the connection between weak genericity and hyperimmunity. Recall that a degree $\mathbf{a}$ is hyperimmune relative to a degree $\mathbf{b}$ if there is an $\mathbf{a}$-computable function that is not majorized by any $\mathbf{b}$-computable function.

Theorem 2.24.12 (Kurtz [228, 229]). *If A is weakly $(n+1)$-generic, then the degree of A is hyperimmune relative to $\mathbf{0^{(n)}}$.*

Proof. Let p_A be the principal function of A. (That is, $p_A(n)$ is the nth element of A in the natural order.) We show that no $\mathbf{0^{(n)}}$-computable function majorizes p_A. Let f be a $\mathbf{0^{(n)}}$-computable function. For a string σ, let p_σ be the principal function of $\{n < |\sigma| : \sigma(n) = 1\}$ (which is a partial function, of course). Let $S_f = \{\sigma : \exists n\, (p_\sigma(n)\downarrow > f(n))\}$. Then S_f is a dense Δ^0_{n+1} set, so A meets S_f, and hence f does not majorize p_A. □

Corollary 2.24.13 (Kurtz [228, 229]). *There is an n-generic degree that is not weakly $(n+1)$-generic.*

Proof. There is an n-generic degree that is below $\mathbf{0^{(n)}}$, and hence not hyperimmune relative to $\mathbf{0^{(n)}}$. □

Theorem 2.24.14 (Kurtz [228, 229]). *A degree $\mathbf{b}$ is the nth jump of a weakly $(n+1)$-generic degree iff $\mathbf{b} > \mathbf{0^{(n)}}$ and $\mathbf{b}$ is hyperimmune relative to $\mathbf{0^{(n)}}$.*

Proof. Suppose that $\mathbf{b} = \mathbf{a^{(n)}}$ with $\mathbf{a}$ weakly $(n+1)$-generic. By Theorem 2.24.12, $\mathbf{a}$ is hyperimmune relative to $\mathbf{0^{(n)}}$, and the class of degrees that are hyperimmune relative to $\mathbf{0^{(n)}}$ is clearly closed upwards. Thus $\mathbf{b}$ is hyperimmune relative to $\mathbf{0^{(n)}}$. Clearly, $\mathbf{b} \geqslant \mathbf{0^{(n)}}$, so in fact $\mathbf{b} > \mathbf{0^{(n)}}$.

The proof of the converse is a variant of the proof of the Friedberg Completeness Criterion (Theorem 2.16.1). Let $\mathbf{b} > \mathbf{0^{(n)}}$ be hyperimmune with respect to $\mathbf{0^{(n)}}$. Let h be a $\mathbf{b}$-computable function that is not majorized by any $\mathbf{0^{(n)}}$-computable function. We may assume that h is increasing and has degree exactly $\mathbf{b}$, and let $B = \operatorname{rng} h$, so that $h = p_B$ and $\deg(B) = \mathbf{b}$. Let $f_0, f_1, \dots$ be uniformly $\mathbf{0^{(n)}}$-computable functions such that $\{\operatorname{rng} f_i : i \in \mathbb{N}\}$ is the collection of all nonempty Σ^0_{n+1} sets of strings. Let $S_i = \operatorname{rng} f_i$ and let $S_i^s = \{\sigma : \exists k \leqslant p_B(s)\, (f_i(k) = \sigma)\}$.

We build a set A by finite extensions $\sigma_0 \preccurlyeq \sigma_1 \preccurlyeq \cdots$ so that $A^{(n)} \equiv_{\mathrm{T}} B$ and A is weakly $(n+1)$-generic. We meet the requirements

$$\mathcal{R}_e : \text{If } S_e \text{ is dense then } A \text{ meets } S_e.$$

We say that $\mathcal{R}_e$ *requires attention* at stage s if σ_s does not meet S_e^s, but there is a string $\tau \succcurlyeq \sigma_s 0 1^{e+1} 0$ that meets S_e^s. Our construction is done relative to B, which can compute $\emptyset^{(n)}$, and hence we can test whether a given requirement requires attention at a given stage.

The peculiar form that τ must have comes from the coding of B into $A^{(n)}$. The number e encodes which requirement is being addressed, which will allow us to "unwind" the construction to recover the coding. We will say that a stage s is a *coding stage* if we act to code information about B into A at stage s. Let $c(s)$ denote the number of coding stages before s.

At stage 0, declare that 0 is a coding stage and let $\sigma_0 = B(0)$.

At stage $s+1$, if no requirement $\mathcal{R}_e$ with $e \leqslant s+1$ requires attention then let $\sigma_{s+1} = \sigma_s$ and declare that $s+1$ is not a coding stage. Otherwise, let $\mathcal{R}_e$ be the strongest priority requirement requiring attention. Let m_0 be the least m such that $f_e(m) \succcurlyeq \sigma_s 0 1^{e+1} 0$. If $|f_e(m_0)| > s+1$, then let $\sigma_{s+1} = \sigma_s$ and declare that $s+1$ is not a coding stage. Otherwise, let $\sigma_{s+1} = f_e(m_0)^\frown B(c(s+1))$ and declare that $s+1$ is a coding stage.

Let $A = \bigcup_s \sigma_s$.

Lemma 2.24.15. *$\mathcal{R}_e$ requires attention only finitely often.*

Proof. Assume by induction that there is a stage s after which no requirement stronger than $\mathcal{R}_e$ requires attention. Suppose $\mathcal{R}_e$ requires attention at a stage $t+1 > s$, and let m_0 be the least m such that $f_e(m) \succcurlyeq \sigma_t 0 1^{e+1} 0$. If $|f_e(m_0)| \leqslant s+1$ then $\mathcal{R}_e$ is met at stage $t+1$ and hence never again requires attention. Otherwise, $\mathcal{R}_e$ continues to be the strongest priority requirement requiring attention until stage $|f_e(m_0)| - 1$, at which point it is met, and never again requires attention. □

Lemma 2.24.16. *A is weakly $(n+1)$-generic.*

Proof. Assume that S_e is dense. Let $g(s)$ be the least k such that for each $\sigma \in 2^{\leqslant s+1}$ there is a $j \leqslant k$ such that $f_e(j)$ extends $\sigma 0 1^{e+1} 0$. Then $g \leqslant_{\mathrm{T}} \emptyset^{(n)}$. Let s be a stage after which no requirement stronger than $\mathcal{R}_e$ ever requires attention. Since p_B is not majorized by any $\emptyset^{(n)}$-computable function, there is a $t > s$ such that $p_B(t) > g(t)$. At stage t, the requirement $\mathcal{R}_e$ requires attention unless it is already met, and continues to be the strongest priority requirement requiring attention from that stage on until it is met. □

Lemma 2.24.17. *$A^{(n)} \leqslant_{\mathrm{T}} B$.*

Proof. Since $\emptyset^{(n)} \leqslant_{\mathrm{T}} B$, the construction of A is computable from B. By the previous lemma, A is weakly $(n+1)$-generic and hence n-generic, so by Theorem 2.24.3, $A^{(n)} \equiv_{\mathrm{T}} A \oplus \emptyset^{(n)} \leqslant_{\mathrm{T}} A \oplus B \equiv_{\mathrm{T}} B$. □

Lemma 2.24.18. *$B \leqslant_{\mathrm{T}} A^{(n)}$.*

Proof. Note that A is infinite, as it is weakly $(n+1)$-generic. Elements are added to A only during coding stages, and there are thus infinitely many such stages $s_0 < s_1 < \cdots$. During stage s_k we define σ_{s_k} so that $\sigma_{s_k}(|\sigma_{s_k}| - 1) = B(k)$. Thus it is enough to show that the function g defined by $g(k) = |\sigma_{s_k}|$ is computable from $A^{(n)}$.

We have $g(0) = 1$. Assume that we have computed $g(k)$ using $A^{(n)}$. We know that $\sigma_{s_{k+1}} \succcurlyeq \sigma_{s_k}01^{e+1}0$ for some e, and we can determine this e from A, since $(A \upharpoonright g(k))^\frown 01^{e+1}0 \prec A$. Since $\emptyset^{(n)} \leqslant A^{(n)}$, we can compute from $A^{(n)}$ the least m such that $f_e(m) \succcurlyeq \sigma_{s_k}01^{e+1}0$, which must exist. By construction, $g(k+1) = |f_e(m)| + 1$. □

Together, the three previous lemmas imply the theorem. □

Taking $n = 0$ in the theorem above, we have the following special case, which will be useful below.

Corollary 2.24.19 (Kurtz [228, 229]). *A degree is weakly 1-generic iff it is hyperimmune.*

Corollary 2.24.20 (Kurtz [228, 229]). *There is a weakly $(n+1)$-generic degree that is not $(n+1)$-generic.*

Proof. By the theorem, there is a weakly $(n+1)$-generic degree $\mathbf{a}$ such that $\mathbf{a}^{(\mathbf{n})} = \mathbf{0}^{(\mathbf{n+1})}$, so that $\mathbf{a}^{(\mathbf{n+1})} = \mathbf{0}^{(\mathbf{n+2})}$. Since $\mathbf{a} < \mathbf{0}^{(\mathbf{n+1})}$, we have $\mathbf{a}^{(\mathbf{n+1})} > \mathbf{a} \vee \mathbf{0}^{(\mathbf{n+1})}$, so by Theorem 2.24.3, $\mathbf{a}$ is not 1-generic. □

Downey, Jockusch, and Stob [121] introduced a new notion of genericity related to array computability. Let $f \leqslant_{\mathrm{pb}} C$ mean that f can be computed from oracle C by a reduction procedure with a primitive recursive bound on the use function. It is easy to show that $f \leqslant_{\mathrm{pb}} \emptyset'$ iff there is a computable function $\widehat{f}(n,s)$ and a primitive recursive function p such that $\lim_s \widehat{f}(n,s) = f(n)$ and $|\{s : \widehat{f}(n,s) \neq \widehat{f}(n,s+1)\}| \leqslant p(n)$. Most proofs yielding results about array noncomputable degrees that use functions $f \leqslant_{\mathrm{wtt}} \emptyset'$ actually use functions $f \leqslant_{\mathrm{pb}} \emptyset'$.

Definition 2.24.21 (Downey, Jockusch, and Stob [121]). (i) A set of strings S is *pb-dense* if there is a function $f \leqslant_{\mathrm{pb}} \emptyset'$ such that $f(\sigma) \succcurlyeq \sigma$ and $f(\sigma) \in S$ for all σ.

(ii) A set is *pb-generic* if it meets every pb-dense set of strings. A degree is pb-generic if it contains a pb-generic set.

For every e, the set $\{\sigma : \sigma \in W_e \vee \forall \tau \succcurlyeq \sigma\,(\tau \notin W_e)\}$ is pb-dense, so every pb-generic set is 1-generic. If $f \leqslant_{\mathrm{pb}} \emptyset'$, then the range of f is Σ^0_2, so every 2-generic set is pb-generic.

The main result about pb-genericity of relevance to us is the following.

Theorem 2.24.22 (Downey, Jockusch, and Stob [121]).

(i) *If* $\mathbf{a}$ *is a.n.c., then there is a pb-generic set* $A \leqslant_{\mathrm{T}} \mathbf{a}$.

(ii) *If* A *is pb-generic, then* $\deg(A)$ *is a.n.c.*

To prove this theorem, we need a lemma. A sequence $\{S_n\}_{n\in\omega}$ of sets of strings is *uniformly wtt-dense* if there is a function $d \leqslant_{\mathrm{wtt}} \emptyset'$ such that $d(n,\sigma) \succcurlyeq \sigma$ and $d(n,\sigma) \in S_n$ for all n and σ.

Lemma 2.24.23 (Downey, Jockusch, and Stob [121]). *If* $\mathbf{a}$ *is a.n.c. and* $\{S_n\}_{n\in\omega}$ *is uniformly wtt-dense, then there is an* $A \leqslant_{\mathrm{T}} \mathbf{a}$ *that meets each* S_n.

Proof. Let $d(n,\sigma)$ witness the uniform wtt-density of the S_n. Let $\widehat{d}$ and b be computable functions such that $\lim_s \widehat{d}(n,\sigma,s) = d(n,\sigma)$ and $|\{s : \widehat{d}(n,\sigma,s) \neq \widehat{d}(n,\sigma,s+1)\}| \leqslant b(n,\sigma)$. We may assume that $\widehat{d}(n,\sigma,s) \succcurlyeq \sigma$ for all σ, n, and s. Let $h(n,\sigma) = \mu s\, \forall t \geqslant s\,(\widehat{d}(n,\sigma,t) = \widehat{d}(n,\sigma,s))$. Let $f(n) = \max\{h(e,\sigma) : e, |\sigma| \leqslant n\}$. We have $f \leqslant_{\mathrm{wtt}} \emptyset'$, since $d \leqslant_{\mathrm{wtt}} \emptyset'$. Fix a function g computable in $\mathbf{a}$ such that $g(n) \geqslant f(n)$ for infinitely many n.

We obtain A as $\bigcup_s \sigma_s$, where $|\sigma_s| = s$. The basic strategy for n at a stage $s > n$ is to let $\sigma_t = \widehat{d}(n,\sigma_s,g(s)) \upharpoonright t$ for all t such that $s < t \leqslant |\widehat{d}(n,\sigma_s,g(s))|$, at which point we declare S_n to be satisfied. If at any stage $u > s$ we find that for some $t \in (g(s),g(u)]$, we have $\widehat{d}(n,\sigma_s,t) \neq \widehat{d}(n,\sigma_s,g(s))$, then we declare S_n to be unsatisfied and repeat the strategy. This strategy can be interrupted at any point by the strategy for an S_m with $m < n$. If this situation happens, we wait until S_m is satisfied and repeat the strategy.

Clearly, $A \leqslant_{\mathrm{T}} \mathbf{a}$, and if S_n is eventually permanently satisfied, then A meets S_n. To see that every S_n is indeed eventually permanently satisfied, assume by induction that there is a least stage $s_0 > n$ by which all S_m with $m < n$ are permanently satisfied. Let $s > s_0$ be such that $g(s) \geqslant f(s)$. If an iteration of the strategy for S_n begins at stage s, then it will clearly lead to S_n becoming permanently satisfied. Otherwise, there must be a stage $v \in [s_0, s)$ such that a strategy for S_n is started at stage v and for all $t \in$

$(g(v), g(s)]$, we have $\widehat{d}(n, \sigma_v, t) = \widehat{d}(n, \sigma_v, g(v))$. But then $g(v) \geqslant h(n, \sigma_v)$, so S_n is again permanently satisfied. $\square$

Proof of Theorem 2.24.22. (i) Let $p_0, p_1, \ldots$ be a uniformly computable listing of the primitive recursive functions. If $\Phi_e^{\emptyset'}(m) \downarrow$ with use at most $p_i(m)$ for all $m \leqslant n$, then let $g(e, i, n) = \Phi_e^{\emptyset'}(n)$. Otherwise, let $g(e, i, n) = 0$. Clearly, $g \leqslant_{\text{wtt}} \emptyset'$ and each function $f \leqslant_{\text{pb}} \emptyset'$ has the form $n \mapsto g(e, i, n)$ for some e and i. Let $\sigma_0, \sigma_1, \ldots$ be an effective listing of $2^{<\omega}$. Let $h(e, i, n) = g(e, i, n)$ if $\sigma_{g(e,i,n)} \supseteq \sigma_n$, and otherwise let $h(e, i, n) = n$. Let $S_{e,i} = \{\sigma_{h(e,i,n)} : n \in \mathbb{N}\}$. Then the sets $S_{e,i}$ are uniformly wtt-dense, and every pb-dense set is equal to $S_{e,i}$ for some e and i. By Lemma 2.24.23, there is a set A computable in $\mathbf{a}$ that meets each $S_{e,i}$, which makes A pb-generic.

(ii) Let $A = \{a_0 < a_1 < \cdots\}$ be pb-generic. Recall that the principal function p_A of A is defined by $p_A(n) = a_n$, and that $m_{\emptyset'}(n)$ is the least s such that $\emptyset' \upharpoonright n = \emptyset'_s \upharpoonright n$. We show that $p_A(n) \geqslant m_{\emptyset'}(n)$ for infinitely many n, which by Theorem 2.23.3 suffices to conclude that $\deg(A)$ is a.n.c. For a string σ, let $i_0 < i_1 < \cdots < i_{j_\sigma - 1}$ be all the numbers such that $\sigma(i_n) = 1$. Let $p_\sigma(n) = i_n$ if $n < j_\sigma$ and let $p_\sigma(n)\uparrow$ otherwise. Fix k. We show that there is an $n \geqslant k$ such that $p_A(n) \geqslant m_{\emptyset'}(n)$. Let

$$S = \{\sigma : \exists n \geqslant k\, (p_\sigma(n) \geqslant m_{\emptyset'}(n))\}.$$

Then S is pb-dense, as witnessed by the function f defined by $f(\sigma) = \sigma 1^k 0^{m_{\emptyset'}(j_\sigma + k)} 1$. Thus A meets S, which means that there is a σ such that $f(\sigma) \prec A$. Then $p_A(j_\sigma + k) \geqslant m_{\emptyset'}(j_\sigma + k)$. $\square$

Since each nonzero c.e. degree bounds a 1-generic degree, but no 2-generic degree is below $\mathbf{0}'$, pb-genericity is an intermediate notion between 1- and 2-genericity.

Corollary 2.24.24 (Downey, Jockusch, and Stob [121]). *There is a* 1-*generic degree that is not pb-generic, and a pb-generic degree that is not* 2-*generic.*