GEMS IN THE FIELD OF BOUNDED QUERIES

William Gasarch
Department of Computer Science
University of Maryland at College Park
MD 20742, USA
gasarch@cs.umd.edu

Abstract Let A be a set. Given $\{x_1, \ldots, x_n\}$, I may want to know (1) which elements of $\{x_1, \ldots, x_n\}$ are in A, (2) how many elements of $\{x_1, \ldots, x_n\}$ are in A, or (3) is $|\{x_1, \ldots, x_n\} \cap A|$ even. All of these can be determined with n queries to A. For which A, n can we get by with fewer queries? Other questions involving 'how many queries do you need to ...' have been posed and (some) answered. This article is a survey of the gems in the field—the results that both answer an interesting question and have a nice proof.

Keywords: Queries, Computability

Introduction

Let the halting problem K be the set of all programs which halt on 0. Assume that I give you 1000 programs and ask you which of them halt, that is, which of them are in K. You cannot answer since K is undecidable. What if I allow you to ask 999 questions to K? Now can you determine which of the 1000 are in K? You can! First build, for each i $0 \leq i \leq 1000$, a program P_i which halts if at least i of the given programs halt. By asking whether $P_i \in K$ you can find out the answer to the query "Do at least i of the given programs halt?" Now binary search allows you to find with 10 queries the number of programs which halt. Say you find out that exactly 783. of the 1000 programs halt. You can run all 1000 of them until you see 783 of them halting, and then you know that the rest do not halt. More generally, if you have $2^n - 1$ programs, you can find out which ones halt by asking n questions. This is the first theorem in the field of Bounded Queries. It was discovered independently by Beigel, Hay, and Owings in the early 1980's.

This observation leads to many other questions of interest. The field of Bounded Queries, founded independently by Beigel [Be87] and Gasarch [Ga85], raises the following types of questions:

(1) Given a function f and a set X, how many queries to X are needed to compute f?

(2) Given a set X and an $n \geq 1$, are there functions that can be computed with n queries to X but not with $n-1$?

This paper is a survey of the nicest results in the field of bounded queries. For a more complete exposition of the field, see [GM99].

1. Definitions

We use notation from [So87], with the notable exception that we use "computable" instead of "recursive" and "c.e." instead of "r.e." This change of terminology was proposed by Soare [So96] for reasons that we agree with; hence we use it. In addition it is being accepted by the community. We remind the reader of some standard notations.

Notation 1.1.

(1) $M_0, M_1, \ldots$ is a list of all Turing machines.

(2) $\varphi_0, \varphi_1, \varphi_2, \ldots$ is a list of all computable partial functions. We obtain this by letting φ_e be the partial function computed by M_e.

(3) $W_0, W_1, \ldots$ is a list of all c.e. sets. We obtain this by letting W_e be the domain of M_e.

(4) $D_0, D_1, \ldots$ is a list of all finite sets indexed in a way that you can effectively recover the elements of the set D_i from the index i. One can view i as a bit vector, so $D_{10111011}$ would represent $\{0, 1, 3, 4, 5, 7\}$. Note that $D_0 = \emptyset$.

(5) $M_0^{()}, M_1^{()}, \ldots$ is a list of all oracle Turing machines.

1.1 Functions of Interest

This paper examines the complexity of the following functions and sets.

Definition 1.2. Let A be a set, and let $n \geq 1$.

(1) $\mathrm{C}_n^A : \mathsf{N}^n \to \{0,1\}^n$ is defined by

$$\mathrm{C}_n^A(x_1, \ldots, x_n) = A(x_1)A(x_2)\cdots A(x_n).$$

(2) $\#_n^A \colon \mathsf{N}^n \to \{0, \ldots, n\}$ is defined by

$$\#_n^A(x_1, \ldots, x_n) = |\{i : x_i \in A\}|.$$

(3) $\mathrm{ODD}_n^A = \{(x_1, \ldots, x_n) \in \mathsf{N}^n : \#_n^A(x_1, \ldots, x_n) \text{ is odd}\}$.

These functions are interesting because they can all be computed with n parallel queries easily; hence the question of whether they can be computed in less than n (perhaps sequential) is intriguing. Also note that C_n^A gives more information then $\#_n^A$ which gives more information then ODD_n^A.

1.2 Bounded Query Classes

Definition 1.3. Let f be a function, $A \subseteq \mathsf{N}$, and $n \in \mathsf{N}$. $f \in \mathrm{FQ}(n, A)$ if $f \leq_{\mathrm{T}} A$ via an algorithm that makes at most n queries to A.

In the introduction we proved that $\mathrm{C}_{2^n-1}^K \in \mathrm{FQ}(n, K)$. Two aspects of that result motivate the next definition.

Aspect 1: When trying to determine $\mathrm{C}_7^K(x_1, \ldots, x_7)$, our first question is "Do at least 4 of the programs halt?" If the answer is YES, we then ask "Do at least 6 of the programs halt?" If the answer to the first question is NO, however, the second question is "Do at least 2 of the programs halt?" Note that the second question asked depends on the answer to the first. Thus the queries are *sequential*. We may want to determine the smallest m such that we can compute $\mathrm{C}_{2^n-1}^K$ with m *parallel* queries to K.

Aspect 2: Let's say that of 7 programs, exactly 3 halt. But suppose that when the queries are made, the answers given are incorrect, so you think there are 4 that halt. If you run them looking for 4 to halt, you will wait forever, so your computation will not terminate. Is there a way to compute $\mathrm{C}_{2^n-1}^K$ with n queries such that even the use of incorrect answers leads to convergence (though perhaps to the wrong bit string)?

Definition 1.4. Let f be a function, $A \subseteq \mathsf{N}$, and $n \in \mathsf{N}$.

(1) $f \in \mathrm{FQ}_{||}(n, A)$ if $f \leq_{\mathrm{T}} A$ via an algorithm that makes at most n queries to A, with the restriction that the queries must be made in *parallel* (i.e., they are nonadaptive). (The symbol $||$ stands for *parallel*.)

(2) $f \in \mathrm{FQC}(n, A)$ if $f \leq_{\mathrm{T}} A$ via an algorithm that makes at most n queries to A, with the restriction that the algorithm must converge (perhaps to the wrong answer) regardless of the choice of oracle (The C stands for *converge*.)

(3) $f \in \mathrm{FQC}_{||}(n, A)$ if $f \leq_{\mathrm{T}} A$ via an algorithm that makes at most n queries to A, with the restrictions that the queries must be made in parallel *and* the algorithm must converge regardless of the choice of oracle.

We now define several bounded-query classes consisting of *sets* that can be decided by making queries to an oracle.

Definition 1.5. Let A, B be sets, and let $n \in \mathsf{N}$.

(1) $B \in \mathrm{Q}(n, A)$ if $\chi_B \in \mathrm{FQ}(n, A)$.

(2) $B \in \mathrm{Q}_{||}(n, A)$ if $\chi_B \in \mathrm{FQ}_{||}(n, A)$.

(3) $B \in \mathrm{QC}(n, A)$ if $\chi_B \in \mathrm{FQC}(n, A)$.

(4) $B \in \mathrm{QC}_{||}(n, A)$ if $\chi_B \in \mathrm{FQC}_{||}(n, A)$.

Definition 1.6. Let f, g be functions. The notions of $f \in \mathrm{FQ}(n, g)$, $f \in \mathrm{FQ}_{||}(n, g)$, etc. can be easily defined.

Definition 1.7. For all the notions in this subsection we can define relativized versions. For example, $\mathrm{FQ}^X(n, A)$ is the set of functions that can be computed with n queries to A and an unlimited number of queries to X.

1.3 Enumerability Classes

The notion of enumerability is very useful in the study of bounded queries. The concept within computability theory is due to Beigel [Be87]. The concept within complexity theory is due independently to Beigel [Be87] and Cai & Hemachandra [CH89]. The term "enumerability" is due to Cai & Hemachandra.

Definition 1.8. Let f be a function, and let $m \geq 1$. We define $f \in \mathrm{EN}(m)$ (f is m-enumerable) in two different ways. We leave it to the reader to show that they are equivalent.

(1) $f \in \mathrm{EN}(m)$ if there exist computable partial functions $g_1, \ldots, g_m$ such that $(\forall x)[f(x) \in \{g_1(x), \ldots, g_m(x)\}]$.

(2) $f \in \mathrm{EN}(m)$ if there is a computable function h such that, for every x, $f(x) \in W_{h(x)}$ and $|W_{h(x)}| \leq m$.

Definition 1.9. Let f be a function, and let $m \geq 1$. We define $f \in \mathrm{SEN}(m)$ (f is strongly m-enumerable) in two different ways. We leave it to the reader to show that they are equivalent.

(1) $f \in \mathrm{SEN}(m)$ if there exist computable functions $g_1, \ldots, g_m$ such that $(\forall x)[f(x) \in \{g_1(x), \ldots, g_m(x)\}]$.

(2) $f \in \mathrm{SEN}(m)$ if there is a computable function h such that, for every x, $f(x) \in D_{h(x)}$ and $|D_{h(x)}| = m$. (One can easily show that the requirements $|D_{h(x)}| \leq m$ and $|D_{h(x)}| = m$ define the same set of functions.)

The following theorem establishes the relationship between query complexity and enumeration complexity.

Theorem 1.10. *Let f be a function, and let $n \in \mathsf{N}$. Then the following statements are equivalent.*

(1) $(\exists X)[f \in \mathrm{FQ}(n, X)]$.

(2) $f \in \mathrm{EN}(2^n)$.

(3) $(\exists Y \equiv_{\mathrm{T}} f)[f \in \mathrm{FQ}_{||}(n, Y) \wedge Y \in \mathrm{Q}(1, f)]$.

Proof: We leave the (easy) proof that (1)⇒(2) to the reader. The fact that (3)⇒(1) is obvious. We present the proof that (2)⇒(3). If $n = 0$, then f is computable, so (2) ⇒(3) is obvious. Hence we assume that $n > 0$.

Suppose that $f \in \mathrm{EN}(2^n)$. We define a set $Y \equiv_{\mathrm{T}} f$ that codes information about f into it so that $f \in \mathrm{FQ}_{||}(n, Y)$; however, Y uses a small amount of information about f so we will have $Y \in \mathrm{Q}(1, f)$.

Assume f is 2^n-enumerable via $g_0, \ldots, g_{2^n-1}$. Let $g_{i,s}(x)$ denote what happens when you run the computation for g_i on input x for s steps. Let t, i be the following functions.

$$t(x) = \mu s[(\exists j)[g_{j,s}(x) \downarrow = f(x)]],$$

$$i(x) = \mu j[g_{j,t(x)}(x) \downarrow = f(x)].$$

We represent $i(x)$ in base 2. We refer to the rightmost bit as the '0th bit', the next bit as the '1st bit', etc. For every k with $0 \leq k \leq n-1$, we define

$$i_k(x) = \text{the } k\text{th bit of } i(x)\ .$$

Let

$$Y = \{(x, k) : i_k(x) = 1\}.$$

It is easy to see that $Y \in \mathrm{Q}(1, f)$ and $f \in \mathrm{FQ}_{||}(n, Y)$ ∎

The next theorem is an analogue of Theorem 1.10 for strong enumeration. We leave the proof to the reader.

Theorem 1.11. *Let f be a function, and let $n \in \mathsf{N}$. Then the following statements are equivalent.*

(1) $(\exists X)[f \in \mathrm{FQC}(n, X)]$.

(2) $f \in \mathrm{SEN}(2^n)$.

(3) $(\exists Y \equiv_{\mathrm{T}} f)[f \in \mathrm{FQC}_{||}(n, Y) \wedge Y \in \mathrm{QC}(1, f)]$.

Note 1.12. All the notions in this subsection can be relativized. For example, $f \in \mathrm{SEN}^X(m)$ if there exist computable-in-X functions $g_1, \ldots, g_m$ such that $(\forall x)[f(x) \in \{g_1(x), \ldots, g_m(x)\}]$.

By Theorems 1.10 and 1.11, any result we obtain about enumerability implies a result about bounded queries. In practice, the results about enumerability are sharper. We state results of both types.

1.4 Definitions from Computability Theory

1.4.1 Selective Sets.

We will use selective sets as defined by Jockusch [Jo68]. (He called them "semirecursive" but he now agrees that "selective" would have been a better name.) These sets have nice properties in terms of bounded queries. In particular, if A is selective, then $(\forall k \geq 1)[\mathrm{C}_k^A \in \mathrm{SEN}(k+1)]$. In Section 4 we will use these sets to help us prove theorems about c.e. sets.

We define selective sets in two ways that are provably equivalent. The equivalence is due to McLaughlin and Appel but was presented in [Jo68].

Definition 1.13. A set A is selective if one of the following two equivalent conditions holds.

(1) There exists a computable function $f\colon \mathsf{N}^2 \to \mathsf{N}$ such that, for all x, y,

- $f(x,y) \in \{x,y\}$, and
- $A \cap \{x,y\} \neq \emptyset \Rightarrow f(x,y) \in A$.

The terminology 'selective set' comes from the fact that f selects which of x, y is more likely to be in A.

(2) There exists a computable linear ordering $\sqsubseteq$ such that A is closed downward under $\sqsubseteq$, i.e., $(\forall x,y)[(x \in A \wedge y \sqsubseteq x) \Rightarrow y \in A]$.

Note 1.14. Let X be a set. We define "selective in X" by making $\sqsubseteq$ and f computable in X in definition 1.13.

Lemma 1.15. *If A is selective, then* $(\forall k \geq 1)[\mathrm{C}_k^A \in \mathrm{SEN}(k+1)]$.

Proof: Let A be selective via ordering $\sqsubseteq$, and let $k \geq 1$. The following algorithm shows that $\mathrm{C}_k^A \in \mathrm{SEN}(k+1)$.

(1) Input $(x_1, \ldots, x_k)$. Renumber so that $x_1 \sqsubseteq \cdots \sqsubseteq x_k$.

(2) Output the set of possibilities $\{1^i 0^{k-i} \mid 0 \leq i \leq k\}$.

∎

1.4.2 Extensive Sets.

We will use extensive sets. These are nice since they are 'almost computable.' In Section 3 we will use them to obtain a lower bound on the enumerability of a function from a lower bound on its strong enumerability. In Section 4 we will use these sets (along with selective sets) to help us prove theorems about c.e. sets.

Definition 1.16. A set X is *extensive* if, for every computable partial function g with finite range, there is a total function $h \leq_{\mathrm{T}} X$ such that h extends g. (The Turing degrees of the extensive sets are the same as the Turing degrees of the consistent extensions of Peano arithmetic [Sc62], [Od89, pages 510–515], but this is not important for our purposes. For this reason they are sometimes called PA sets. They have also been referred to as DNR_2 sets, which stands for Diagonally Non Recursive; see [Jo89].)

Note 1.17. Clearly K is extensive. It is easy to show that all extensive sets are not computable. What is of more interest, although we will not use it, is that there are low extensive sets [JS72].

Jockusch and Soare [JS72] proved the following theorem. We will use it in Theorems 3.4 and 4.4 to show that a set A is computable by showing that A is computable in *every* extensive set.

Theorem 1.18. *There exists a minimal pair of extensive sets. That is, there exist extensive sets X_1 and X_2 such that, for all A, if $A \leq_{\mathrm{T}} X_1$ and $A \leq_{\mathrm{T}} X_2$ then A is computable.*

1.4.3 Computably Bounded Sets.

We will use computably bounded sets. These are nice in terms of convergence (see Lemma 1.20 below) and hence will be used when we study the difference between $\mathrm{FQ}(n, A)$ and $\mathrm{FQC}(n, A)$ (also between $\mathrm{Q}(n, A)$ and $\mathrm{QC}(n, A)$).

Definition 1.19. A set X is *computably bounded* (abbreviated c.b.) if, for every (total) function $f : \mathsf{N} \to \mathsf{N}$ such that $f \leq_{\mathrm{T}} X$, there exists a computable function g such that $(\forall x)[f(x) < g(x)]$. (In the literature, the Turing degrees of c.b. sets are said to be *hyperimmune free*; this term comes from an equivalent definition that we are not using.)

The following theorem is due to Miller and Martin [MM68]; the proof can also be found in [Od89]. In Section 5.2 we will use this theorem to explore questions about more queries being more powerful.

Lemma 1.20.

(1) There exist c.b. sets $B \leq_{\mathrm{T}} \emptyset''$.

(2) If B is a c.b. set and $A \leq_{\mathrm{T}} B$, then $A \leq_{\mathrm{tt}} B$.

2. The Complexity of C_n^A

The function $\mathrm{C}_n^A(x_1, \ldots, x_n) = A(x_1) \cdots A(x_n)$ can clearly be computed with n queries to A and is clearly 2^n-enumerable. We have already seen that neither result is tight, as C_n^K can be computed with $O(\log n)$ queries to K and C_n^K is $(n+1)$-enumerable.

Is C_n^K n-enumerable? What about sets other than K? This section will address these and other questions.

2.1 The Complexity of C_n^A for General Sets

In this section we show that, for every set A, if $(\exists n)(\exists X)[\mathrm{C}_{2^n}^A \in \mathrm{FQ}(n, X)]$ then A is computable. By Theorem 1.10, it suffices to show that if $(\exists n)[\mathrm{C}_{2^n}^A \in \mathrm{EN}(2^n)]$ then A is computable. Note that 2^n appears in two places. It is cleaner to prove the stronger statement that if $(\exists n)[\mathrm{C}_n^A \in \mathrm{EN}(n)]$ then A is computable.

Recall that in the introduction we answered 1000 instances of K by asking 10 *sequential* queries to K. Corollary 2.4 below will show that the questions *need* to be sequential; that is, 999 *parallel* queries would not suffice.

The following theorem was first proved by Beigel in his thesis [Be87]. It also appears in [BGGO93].

Definition 2.1. If $1 \leq i \leq j \leq n$ and $b = b_1 b_2 \cdots b_n \in \{0,1\}^n$, then $b[i:j]$ is defined as $b_i b_{i+1} \cdots b_j$.

Theorem 2.2. *Let $m \geq 1$, and let A be a set such that $\mathrm{C}_m^A \in \mathrm{EN}(m)$. Then A is computable.*

Proof: The proof is by induction on m. The conclusion of the theorem is obvious if $m = 1$; hence the base case is established. So assume that $m > 1$, and that the $m-1$ case is true. We show that A is computable.

Assume $\mathrm{C}_m^A \in \mathrm{EN}(m)$ via $g_1, \ldots, g_m$. We would *like* to show that $\mathrm{C}_{m-1}^A \in \mathrm{EN}(m-1)$; by the induction hypothesis, this would prove that A is computable. So we attempt to show the existence of computable partial functions $h_1, \ldots, h_{m-1}$ such that $\mathrm{C}_{m-1}^A \in \mathrm{EN}(m-1)$ via $h_1, \ldots, h_{m-1}$. Either our algorithm, A1, works, or its very failure to do so leads to an algorithm, A2, that decides A outright.

We have $\mathrm{C}_m^A \in \mathrm{EN}(m)$. We want $\mathrm{C}_{m-1}^A \in \mathrm{EN}(m-1)$. Given $(x_1, \ldots, x_{m-1})$ we want to somehow use the $\mathrm{C}_m^A \in \mathrm{EN}(m)$ algorithm. We will do this by adding to $(x_1, \ldots, x_{m-1})$ a variety of y's to form m elements and then running the $\mathrm{C}_m^A \in \mathrm{EN}(m)$ algorithm. The hope is to find a y such that we know

$$|\{g_1(x_1, \ldots, x_{m-1}, y)[1:m-1], \ldots, g_m(x_1, \ldots, x_{m-1}, y)[1:m-1]\}| \leq m-1.$$

ALGORITHM A1

(1) Input $(x_1, \ldots, x_{m-1})$.

(2) Search for i, j, y such that $1 \leq i < j \leq m$, $g_i(x_1, \ldots, x_{m-1}, y) \downarrow$, $g_j(x_1, \ldots, x_{m-1}, y) \downarrow$, and

$$g_i(x_1, \ldots, x_{m-1}, y)[1 : m-1] = g_j(x_1, \ldots, x_{m-1}, y)[1 : m-1].$$

(3) (If this step is reached, then i, j, and y were found in step 2.) For every l with $1 \leq l \leq m-1$, let

$$h_l(x_1, \ldots, x_{m-1}) = \begin{cases} g_l(x_1, \ldots, x_{m-1}, y)[1 : m-1], & \text{if } l < j; \\ g_{l+1}(x_1, \ldots, x_{m-1}, y)[1 : m-1], & \text{if } l \geq j. \end{cases}$$

Let $x_1, \ldots, x_{m-1} \in \mathsf{N}$. We show that *if* $\mathrm{A1}(x_1, \ldots, x_{m-1}) \downarrow$, *then*

$$\mathrm{C}^A_{m-1}(x_1, \ldots, x_{m-1}) \in \{h_l(x_1, \ldots, x_{m-1}) : 1 \leq l \leq m-1\}.$$

So suppose that $\mathrm{A1}(x_1, \ldots, x_{m-1}) \downarrow$. Since step 2 terminates, we have i, j, y such that $1 \leq i < j \leq m$, $g_i(x_1, \ldots, x_{m-1}, y) \downarrow$, $g_j(x_1, \ldots, x_{m-1}, y) \downarrow$, and

$$g_i(x_1, \ldots, x_{m-1}, y)[1 : m-1] = g_j(x_1, \ldots, x_{m-1}, y)[1 : m-1].$$

Thus the set

$$\{g_1(x_1, \ldots, x_{m-1}, y)[1 : m-1], \ldots, g_m(x_1, \ldots, x_{m-1}, y)[1 : m-1]\}$$

has at most $m-1$ elements, and is equal to the set

$$\{h_1(x_1, \ldots, x_{m-1}), \ldots, h_{m-1}(x_1, \ldots, x_{m-1})\}.$$

Moreover, $\mathrm{C}^A_{m-1}(x_1, \ldots, x_{m-1})$ is in this set, by our assumption about A and our choice of $g_1, \ldots, g_m$.

We show that either algorithm A1 yields $\mathrm{C}^A_{m-1} \in \mathrm{EN}(m-1)$, or some other algorithm (A2, built out of the failure of A1 to work) yields that A is computable.

Case 1: $(\forall x_1, \ldots, x_{m-1})[\mathrm{A1}(x_1, \ldots, x_{m-1}) \downarrow]$. Then by the reasoning above, $\mathrm{C}^A_{m-1} \in \mathrm{EN}(m-1)$ via $h_1, \ldots, h_{m-1}$. By the induction hypothesis, A is computable.

Case 2: $(\exists x'_1, \ldots, x'_{m-1})[A1(x'_1, \ldots, x'_{m-1}) \uparrow]$. We use this tuple to devise a new algorithm, A2, that shows outright that A is computable.

Since $\mathrm{A1}(x'_1, \ldots, x'_{m-1}) \uparrow$, note that, for all $i < j \leq m$ and for every y, it cannot be the case that $g_i(x'_1, \ldots, x'_{m-1}, y)$ and $g_j(x'_1, \ldots, x'_{m-1}, y)$ converge and their outputs agree on the first $m-1$ bits. Let $b'_1 \cdots b'_{m-1} = \mathrm{C}^A_{m-1}(x'_1, \ldots, x'_{m-1})$.

ALGORITHM A2

(1) Input y.

(2) Dovetail the $g_1(x'_1, \ldots, x'_{m-1}, y), \ldots, g_m(x'_1, \ldots, x'_{m-1}, y)$ computations, stopping when you find i and a bit b such that

$$g_i(x'_1, \ldots, x'_{m-1}, y) \downarrow = b'_1 \cdots b'_{m-1} b.$$

(3) (If this step is reached, then i and b were found in step 2.) Output b.

Let $y \in \mathsf{N}$. We show that $\mathrm{A2}(y) \downarrow = A(y)$. Since

$$\mathrm{C}^A_m(x'_1, \ldots, x'_{m-1}, y) \in \{g_1(x'_1, \ldots, x'_{m-1}, y), \ldots, g_m(x'_1, \ldots, x'_{m-1}, y)\},$$

we know that $(\exists i, b)[g_i(x'_1, \ldots, x'_{m-1}, y) \downarrow = b'_1 \cdots b'_{m-1} b]$. Hence $\mathrm{A2}(y) \downarrow$. If in step 2 it is discovered that $g_i(x'_1, \ldots, x'_{m-1}, y) \downarrow = b'_1 \cdots b'_{m-1} b$, then it cannot be the case that $(\exists j \neq i)(\exists b')[g_j(x'_1, \ldots, x'_{m-1}, y) \downarrow = b'_1 \cdots b'_{m-1} b']$, since this would imply that

$$g_i(x'_1, \ldots, x'_{m-1}, y)[1 : m-1] = g_j(x'_1, \ldots, x'_{m-1}, y)[1 : m-1],$$

contrary to the choice of $x'_1, \ldots, x'_{m-1}$. Hence for every $j \neq i$, either $g_j(x'_1, \ldots, x'_{m-1}, y)$ diverges, or it converges and is wrong on one of the first $m-1$ bits. It follows that $g_i(x'_1, \ldots, x'_{m-1}, y) = \mathrm{C}^A_m(x'_1, \ldots, x'_{m-1}, y)$, so $A(y) = b = \mathrm{A2}(y)$. ∎

Corollary 2.3. *Let $n \in \mathsf{N}$ and $A, X \subseteq \mathsf{N}$. If $\mathrm{C}^A_{2^n} \in \mathrm{FQ}(n, X)$, then A is computable.*

Proof: This follows from Theorems 2.2 and 1.10. ∎

This survey began by showing that you could answer 1000 queries to K with 10 *sequential* queries to K. By this next corollary we know that the sequential nature is inherent– we could not have answered 1000 queries to K with 999 *parallel* queries to K.

Corollary 2.4. *Let $n \geq 1$ and $A \subseteq \mathsf{N}$. If $\mathrm{C}^A_n \in \mathrm{FQ}_{||}(n-1, A)$, then A is computable.*

Proof: If $\mathrm{C}^A_n \in \mathrm{FQ}_{||}(n-1, A)$ then an easy induction shows that, for all $m \geq n$, $\mathrm{C}^A_m \in \mathrm{FQ}_{||}(n-1, A)$. In particular,

$$\mathrm{C}^A_{2^{n-1}} \in \mathrm{FQ}_{||}(n-1, A) \subseteq \mathrm{FQ}(n-1, A).$$

By Theorem 1.10, $\mathrm{FQ}(n-1, A) \subseteq \mathrm{EN}(2^{n-1})$, so we have $\mathrm{C}^A_{2^{n-1}} \in \mathrm{EN}(2^{n-1})$. By Theorem 2.2, A is computable. ∎

The queries in the algorithm given in the introduction (the one that showed that $\mathrm{C}^K_{2^n-1} \in \mathrm{FQ}(n, K)$) were made sequentially, and now we know that this is inherent. What if we use another oracle? By Theorem 1.10, there exists a Y such that $\mathrm{C}^K_{2^n-1} \in \mathrm{FQ}_{||}(n, Y)$. Note that the set Y is useful *not* because it has high Turing degree (in fact, $K \equiv_{\mathrm{T}} Y$) but because of the way information is stored in it.

2.2 The Complexity of C^B_n for Natural Sets B

Sets that are Σ_i-complete or Π_i-complete are natural. The set K is surely natural, and we have $\mathrm{C}^K_n \in \mathrm{EN}(n+1)$. Are there other natural sets B for which $\mathrm{C}^B_n \in \mathrm{EN}(n+1)$, or at least $\mathrm{C}^B_n \in \mathrm{EN}(2^n-1)$. The answer is NO. We show that for every noncomputable set A, $\mathrm{C}^{A'}_n \notin \mathrm{EN}(2^n-1)$. This result first appeared in [BGGO93].

Definition 2.5. Let $k, n \in \mathsf{N}$ such that $1 \leq k \leq n$. $S(n,k) = \sum_{i=0}^{k-1} \binom{n}{i}$.

The following lemma has appeared in several places independently. It was first discovered by Vapnik and Chervonenkis [VC71], and subsequently rediscovered by Sauer [Sa72], Clarke, Owings, and Spriggs [COS75], and Beigel [Be87]. The reason why it has been discovered by so many is that it has applications in probability theory [VC71], computational learning theory [BEHW89], computational geometry [HW87], and of course bounded queries. It has also been attributed to Shelah [Sh72]; however, that paper does not contain it. I suspect that Shelah had a proof, and that people refer to that paper because its title *sounds* as if it should contain it.

Lemma 2.6. *Let $Y \subseteq \{0,1\}^n$. Let $k \leq n$. Assume that for every $i_1, .., i_k$ with $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, the projected set*

$$\{\vec{b'} \in \{0,1\}^k \mid (\exists \vec{b} \in Y)[\vec{b'} \text{ is the projection of } \vec{b} \text{ on coordinates } i_1, \ldots, i_k\,]\}$$

has at most $2^k - 1$ elements. Then Y has at most $S(n,k)$ elements.

The next lemma shows that if you can save just a little bit on enumerabiliy (that is, $\mathrm{C}^B_k \in \mathrm{EN}(2^k-1)$ instead of the obvious $\mathrm{C}^B_k \in \mathrm{EN}(2^k)$), then, for large n, you can save a lot in terms of enumerability. This will enable us to show that for the jump of any noncomputable set, you cannot even save a little.

Theorem 2.7. *Let $k \geq 1$, and let B be a set. If $\mathrm{C}^B_k \in \mathrm{EN}(2^k-1)$, then $(\forall n \geq k)[\mathrm{C}^B_n \in \mathrm{EN}(S(n,k))]$. If $\mathrm{C}^B_k \in \mathrm{SEN}(2^k-1)$, then $(\forall n \geq k)[\mathrm{C}^B_n \in \mathrm{SEN}(S(n,k))]$.*

Proof: Assume that $\mathrm{C}^B_k \in \mathrm{EN}(2^k-1)$ via g. The proof for $\mathrm{SEN}(2^k-1)$ is similar.

We show $\mathrm{C}^B_n \in \mathrm{EN}(S(n,k))$. In input $(x_1, \ldots, x_n)$ enumerate elements of $\{0,1\}^n$ as follows. Enumerate all strings $b_1 \cdots b_n$ such that, for all $i_1, \ldots, i_k$

with $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, the projection $b_{i_1} b_{i_2} \cdots b_{i_k}$ shows up in $W_{g(x_{i_1}, x_{i_2}, \ldots, x_{i_k})}$

Let Y be the set of strings enumerated. We need to show that $\mathrm{C}_n^B \in Y$ and that $|Y| \leq S(n,k)$.

Let $\mathrm{C}_n^B = \vec{b}$. For every $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ we clearly have the projection $b_{i_1} b_{i_2} \cdots b_{i_k}$ in $W_{g(x_{i_1}, x_{i_2}, \ldots, x_{i_k})}$; hence $\vec{b} \in Y$.

For every $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ the number of elements in Y projected on those coordinates is at most $2^k - 1$ since $|W_{g(x_{i_1}, x_{i_2}, \ldots, x_{i_k})}| \leq 2^k - 1$; hence by Lemma 2.6 $|Y| \leq S(n,k)$. ∎

Theorem 2.8. *If* $(\exists k \geq 1)[\mathrm{C}_k^{A'} \in \mathrm{EN}(2^k - 1)]$, *then* A *is computable*.

Proof: Assume $(\exists k \geq 1)[\mathrm{C}_k^{A'} \in \mathrm{EN}(2^k - 1)]$. We show that $(\exists n \geq 1)[\mathrm{C}_n^A \in \mathrm{EN}(n)]$, hence that A is computable (by Theorem 2.2).

By Theorem 2.7, $(\forall n \geq k)[\mathrm{C}_n^{A'} \in \mathrm{EN}(S(n,k))]$. For large n, $S(n,k) = O(n^k)$, hence we write $\mathrm{C}_n^{A'} \in \mathrm{EN}(O(n^k))$.

Since $A \leq_m A'$, we have $\mathrm{C}_n^A \in \mathrm{EN}(O(n^k))$. By Theorem 1.10,

$$(\exists Y \equiv_{\mathrm{T}} A)[\mathrm{C}_n^A \in \mathrm{FQ}_{||}(O(k \log n), Y) = \mathrm{FQ}_{||}(O(\log n), Y) = \mathrm{FQ}(1, \mathrm{C}_{O(\log n)}^Y)].$$

Since $Y \equiv_{\mathrm{T}} A$, we have $Y \leq_m A'$, hence

$$\mathrm{C}_n^A \in \mathrm{FQ}(1, \mathrm{C}_{O(\log n)}^Y) \subseteq \mathrm{FQ}(1, \mathrm{C}_{O(\log n)}^{A'}).$$

By Theorem 2.7, $\mathrm{C}_{O(\log n)}^{A'} \in \mathrm{EN}(S(O(\log n), k)) \subseteq \mathrm{EN}(O((\log n)^k))$. Hence $\mathrm{C}_n^A \in \mathrm{EN}(O((\log n)^k))$. For n large, $O((\log n)^k) \leq n$, which implies that $\mathrm{C}_n^A \in \mathrm{EN}(n)$. ∎

By a proof similar to that of Theorem 2.7, we obtain the following.

Theorem 2.9. *Let* $k \geq 1$, *and let* A *be a set. If* $\mathrm{C}_k^A \in \mathrm{SEN}(2^k - 1)$, *then* $(\forall n \geq k)[\mathrm{C}_n^A \in \mathrm{SEN}(S(n,k))]$.

3. The Complexity of $\#_n^A$

We now know that for noncomputable sets A, $\mathrm{C}_n^A \notin \mathrm{EN}(n)$. What if we ask for less information? Realize that there are 2^n possibilities for C_n^A, which is a lot. In contrast, there are only $n+1$ possibilities for $\#_n^A$, so perhaps there are some noncomputable sets A such that $\#_n^A \in \mathrm{EN}(n)$. Alas, no such set exists!

Beigel conjectured that if $\#_n^A \in \mathrm{EN}(n)$, then A is computable. Owings [Ow89] showed that if $\#_n^A \in \mathrm{SEN}(n)$, then A is computable, and also that if $\#_2^A \in \mathrm{EN}(1)$, then A is computable. (Owings stated his theorem as $\#_n^A \in \mathrm{EN}(n) \Rightarrow A \leq_{\mathrm{T}} K$. The form we state follows from the same proof.)

Kummer [Ku92] then proved Beigel's conjecture. Kummer's proof used a Ramsey-type theorem on trees. Later, Kummer and Stephan [KS94] observed that Owings' proof could be extended to show Beigel's conjecture. We present Owings' proof followed by Kummer and Stephan's observation.

Most theorems in computability theory relativize. For the next theorem, we *need* to prove its relativized form, since we need this stronger version as a way to strengthen the induction hypothesis.

Notation 3.1. If D is a set, then $\mathcal{P}(D)$ denotes the power set of D.

If $\#_n^A \in \mathrm{SEN}^X(n)$ via f then $f \leq_{\mathrm{T}} X$ and $f(x_1, \ldots, x_n)$, outputs $\leq n$ possibilities for $\#_n^A(x_1, \ldots, x_n)$. Note that there may be other sets B such that $\#_n^B \in \mathrm{SEN}^X(n)$ via f. We will be interested in the set of all such sets. The next definition clarifies these concepts.

Definition 3.2. Let X be a set, let $n \geq 1$, and let f be a function such that

- $f\colon \mathsf{N}^n \to \mathcal{P}(\{0, \ldots, n\})$,
- $f \leq_{\mathrm{T}} X$, and
- $(\forall x_1, \ldots, x_n)[|f(x_1, \ldots, x_n)| \leq n]$.

The triple (n, f, X) is *helpful*, and

$$SE_f = \{Z \mid (\forall x_1, \ldots, x_n)[\#_n^Z(x_1, \ldots, x_n) \in f(x_1, \ldots, x_n)]\}.$$

(Note that the sets in SE_f are precisely the sets Z for which f is a strong enumerator-in-X of $\#_n^Z$.)

Theorem 3.3. *Let $n \geq 1$, and let A, X be sets such that $\#_n^A \in \mathrm{SEN}^X(n)$. Then $A \leq_{\mathrm{T}} X$.*

Proof: Since $\#_n^A \in \mathrm{SEN}^X(n)$, there exists a function $f\colon \mathsf{N}^n \to \mathcal{P}(\{0, \ldots, n\})$ such that (n, f, X) is helpful and $A \in SE_f$.

We show that since (n, f, X) is helpful, we have $(\forall C \in SE_f)[C \leq_{\mathrm{T}} X]$. Our proof is by induction on n. For $n = 1$, SE_f has only one element, which is clearly computable in X.

So assume that $n \geq 2$ and, as induction hypothesis, that for every set Y and every function $g\colon \mathsf{N}^{n-1} \to \mathcal{P}(\{0, \ldots, n-1\})$,

$$(n-1, g, Y) \text{ helpful} \Rightarrow (\forall D \in SE_g)[D \leq_{\mathrm{T}} Y].$$

To prove that $A \leq_{\mathrm{T}} X$, we actually prove that the following three conditions hold.

(1) $(\forall B, C \in SE_f)[C \leq_{\mathrm{T}} B \oplus X]$.

(2) SE_f is countable.

(3) $(\forall C \in SE_f)[C \leq_{\mathrm{T}} X]$.

That (1) $\Rightarrow$ (2) is trivial. That (2) $\Rightarrow$ (3) comes from the following two facts: (a) SE_f is the set of infinite branches of a tree that is computable in f, (b) any tree with countably many infinite branches (but at least one) has some branch computable in the tree (proved by Owings, and independently by Jockusch and Soare [JS72]; see also [Od89, Prop. V.5.27, page 507]).

Hence we need prove only (1).

Let $B, C \in SE_f$. To prove that $C \leq_{\mathrm{T}} B \oplus X$, we first show that $B - C \leq_{\mathrm{T}} B \oplus X$ and $C - B \leq_{\mathrm{T}} B \oplus X$.

To show that $B - C \leq_{\mathrm{T}} B \oplus X$, we use the induction hypothesis. In particular, we show that there exists an $(n-1)$-ary function g such that $(n-1, g, B \oplus X)$ is helpful and $B - C \in SE_g$. By the induction hypothesis, this yields $B - C \leq_{\mathrm{T}} B \oplus X$.

If $B - C = \emptyset$, then clearly $B - C \leq_{\mathrm{T}} B \oplus X$ and we are done. Hence we can assume $B - C \neq \emptyset$, so choose $z_0 \in B - C$. We use z_0 in our algorithm for g.

ALGORITHM FOR g

(1) Input $(x_1, \ldots, x_{n-1})$.

(2) Make the queries "$x_1 \in B$?", ..., "$x_{n-1} \in B$?". If there is some i such that $x_i \notin B$, then $x_i \notin B - C$; hence $\#_{n-1}^{B-C}(x_1, \ldots, x_{n-1}) < n-1$, so let $g(x_1, \ldots, x_{n-1}) = \{0, \ldots, n-2\}$ and halt. Otherwise, go to the next step.

(3) (Since we have reached this step, we know that $\{x_1, \ldots, x_{n-1}\} \subseteq B$.) Note that $B - C \cap \{x_1, \ldots, x_{n-1}\} = \overline{C} \cap \{x_1, \ldots, x_{n-1}\}$. Hence $\#_{n-1}^{B-C}(x_1, \ldots, x_{n-1}) = n - 1 - \#_{n-1}^{C}(x_1, \ldots, x_{n-1})$. Therefore, we need only find $\leq n-1$ possibilities for $\#_{n-1}^{C}(x_1, \ldots, x_{n-1})$.

(4) Since $z_0 \notin C$, we have $\#_{n-1}^{C}(x_1, \ldots, x_{n-1}) = \#_{n}^{C}(x_1, \ldots, x_{n-1}, z_0)$. Therefore, we need only find $\leq n-1$ possibilities for $\#_{n}^{C}(x_1, \ldots, x_{n-1}, z_0)$.

(5) Since $x_1, \ldots, x_{n-1}, z_0 \in B$ and $B \in SE_f$, we have $n \in f(x_1, \ldots, x_{n-1}, z_0)$. Since $z_0 \notin C$, we know that $\#_{n}^{C}(x_1, \ldots, x_{n-1}, z_0) \neq n$. Moreover, $C \in SE_f$, so $\#_{n}^{C}(x_1, \ldots, x_{n-1}, z_0) \in f(x_1, \ldots, x_{n-1}, z_0) - \{n\}$. Now note that $|f(x_1, \ldots, x_{n-1}, z_0) - \{n\}| \leq n-1$, so we have $\leq n-1$ possibilities for $\#_{n}^{C}(x_1, \ldots, x_{n-1}, z_0)$. Using this information, define $g(x_1, \ldots, x_{n-1})$ and halt.

Since (n, f, X) is helpful (hence $f \leq_{\mathrm{T}} X$) and $B, C \in SE_f$, it is clear from the algorithm that $(n-1, g, B \oplus X)$ is helpful and $B - C \in SE_g$. By the induction hypothesis, $B - C \leq_{\mathrm{T}} B \oplus X$.

Noting that $C-B = \overline{B}-\overline{C}$, we easily see that the proof that $C-B \leq_{\mathrm{T}} B\oplus X$ is similar. Now

$$C = (C \cap B) \cup (C - B) = [\overline{(B - C)} \cap B] \cup (C - B).$$

Since we have shown that both $B - C$ and $C - B$ are computable in $B \oplus X$, we have $C \leq_{\mathrm{T}} B \oplus X$. This proves that condition (1) holds. Since $(1) \Rightarrow (3)$, we also have $C \leq_{\mathrm{T}} X$. ∎

By Theorem 3.3, we have that if $\#_n^A \in \mathrm{SEN}(n)$, then A is computable. We want to obtain the analogous result for $\mathrm{EN}(n)$. Extensive sets X are used, since they can turn an $\mathrm{EN}(n)$ computation into an $\mathrm{SEN}^X(n)$ computation.

Theorem 3.4. *Let $n \geq 1$, and let A be a set. If $\#_n^A \in \mathrm{EN}(n)$, then A is computable.*

Proof: Assume $\#_n^A$ is n-enumerable via computable partial functions $h_1, \ldots, h_n$. Thus for all $x_1, \ldots, x_n \in \mathsf{N}$,

$$\#_n^A(x_1, \ldots, x_n) \in \{h_1(x_1, \ldots, x_n), \ldots, h_n(x_1, \ldots, x_n)\}.$$

We can assume that each h_i has finite range, namely $\{0, \ldots, n\}$.

Let X be *any* extensive set (see Definition 1.16). Since X is extensive, each h_i has a total extension that is computable in X. Using these extensions, we obtain that $\#_n^A \in \mathrm{SEN}^X(n)$. By Theorem 3.3, we have $A \leq_{\mathrm{T}} X$. Since X was *any* extensive set, we have that, for *every* extensive set X, $A \leq_{\mathrm{T}} X$. Since there exist minimal pairs of extensive sets (Theorem 1.18), we obtain that A is computable. ∎

4. The Complexity of ODD_n^A

We now know that if A is noncomputable, then $\#_n^A \notin \mathrm{EN}(n)$. Asking for $\#_n^A$ seems (in retrospect) like asking for a lot of information. What if we just want to know the parity of $\#_n^A$? Determining the parity of $\#_n^A$ entails finding only one bit of information. This problem is easy in terms of enumerability in a trivial way: there are only two possibilities for it.

If we return to bounded queries (rather than enumerability), interesting questions arise. For example, how hard is ODD_n^A in terms of queries to A?

The main theorem of this chapter is the following:

If A is c.e. and $(\exists n \geq 1)[\mathrm{ODD}_n^A \in \mathrm{Q}_{||}(n-1, A)]$, then A is computable.

We will give two proofs of this theorem. The first proof shows that if $\mathrm{ODD}_n^A \in \mathrm{Q}_{||}(n-1, A)$ and A is c.e. then $\mathrm{C}_n^A \in \mathrm{EN}(n)$, hence by Theorem 2.2, A is computable. This proof gives intuition for the result but involves some details that need to be done carefully. The second proof uses selective

and extensive sets and is very elegant; however, it is less insightful as to why the theorem is true. I leave as an exercise the task of comparing the two proofs and determining which one is better.

The following are also known.

(1) If A and B are c.e. and $\text{ODD}_n^A \in \text{Q}_{||}(n-1, B)$ then A is computable.

(2) If A and B are c.e. and $\text{ODD}_{2^n}^A \in \text{Q}(n, B)$ then A is computable.

The first result can be proven by simple variations of the proofs given here. The second one is more complicated. The results in this section, and the two results stated below that we are not going to prove, have appeared in both [GM99] and [Be*etal* 00]. Our main result has three proofs. We present two here; one additional proof can be found in [GM99].

4.1 A Direct Proof

Theorem 4.1. *If A is computably enumerable and $\text{ODD}_n^A \in \text{Q}_{||}(n-1, A)$ then A is computable.*

Proof: In the following we assume that A has an enumeration A_s and that ODD_n^A is computed by M with $n-1$ parallel queries to A itself. We show that then C_n^A is in $\text{EN}(n)$ which gives that A is computable by Theorem 2.2.
ALGORITHM FOR $\text{C}_n^A \in \text{EN}(n)$.

(1) Input $(x_1, \ldots, x_n)$.

(2) Run $M^{()}(x_1, \ldots, x_n)$ until the queries are made. Let them be $(y_1, \ldots, y_{n-1})$. (Do not ask them.)

(3) Enumerate a tuple $(A_s(x_1), \ldots, A_s(x_n))$ as a possiblity for C_n^A iff the computation $M(x_1, \ldots, x_n)$ with query answers $(A_s(y_1), \ldots, A_s(y_{n-1}))$ terminates within s steps and its output agrees with $A_s(x_1) + \ldots + A_s(x_n)$ modulo 2.

The enumerated set contains $\text{C}_n^A(x_1, \ldots, x_n)$ since for sufficiently large s the sets A_s and A coincide at all queried places and M has converged. We now show that we have enumerated at most n strings. There are two cases.
Case 1: There are two different outputs $(a_1, \ldots, a_n)$ and later $(b_1, \ldots, b_n)$ where the corresponding computation $M(x_1, \ldots, x_n)$ with query answers $(A_s(y_1), \ldots, A_s(y_n)))$ uses in both cases the same values $c_1, \ldots, c_{n-1}$ for $A_s(y_1), \ldots, A_s(y_{n-1})$. Then $a_1 + a_2 + \ldots + a_n + 2 \leq b_1 + b_2 + \ldots + b_n$ and no tuple with cardinality $a_1 + a + 2 + \ldots + a_n + 1$ is enumerated. Since for every cardinality at most one tuple is enumerated, at most n elements are enumerated.
Case 2: For every output $(a_1, \ldots, a_n)$ the corresponding computation $M(x_1, \ldots, x_n)$ with query answers $(A_s(y_1), \ldots, A_s(y_n))$ uses values $c_1, \ldots, c_{n-1}$ for $A_s(y_1), \ldots, A_s(y_{n-1})$

not used for any other output. As these values $c_1, \ldots, c_{n-1}$ originate from an enumeration of A, this tuple can take at most n values and so the cardinality of the set enumerated is at most n. ∎

4.2 An Elegant Proof

We show the following.

- If A is selective and $(\exists n \geq 1)[\mathrm{ODD}_n^A \in \mathrm{Q}_{||}(n-1, A)]$, then A is computable.
- If A is c.e. and $(\exists n \geq 1)[\mathrm{ODD}_n^A \in \mathrm{Q}_{||}(n-1, A)]$, then A is computable.

In investigating the complexity of ODD_n^A, we first look at selective sets A, and we then proceed to c.e. sets A. This is because we obtain the result about c.e. sets *from* the result about selective sets.

4.2.1 ODD_n^A for Selective Sets A.

Theorem 4.2. *Let $n \geq 1$, and let A be a selective set such that $\mathrm{ODD}_n^A \in \mathrm{Q}_{||}(n-1, A)$. Then A is computable.*

Proof: If $n = 1$, then since $(\forall x)[\mathrm{ODD}_1^A(x) = A(x)]$, we have that A is computable. Hence we can assume that $n \geq 2$. Let A be selective via $\sqsubseteq$, and assume that $\mathrm{ODD}_n^A \in \mathrm{Q}_{||}(n-1, A)$ via M^A.

The following algorithm shows that $\mathrm{C}_{2n+1}^A \in \mathrm{FQ}_{||}(2n, A)$. By Corollary 2.4, this yields that A is computable.

(1) Input $(x_1, \ldots, x_{2n+1})$, where $x_1 \sqsubseteq \cdots \sqsubseteq x_{2n+1}$. Note that

$$\mathrm{C}_{2n+1}^A(x_1, \ldots, x_{2n+1}) \in \{1^i 0^{2n+1-i} \mid 0 \leq i \leq 2n+1\}.$$

(2) Simulate the computation of $M^A(x_2, x_4, x_6, \ldots, x_{2n})$ to obtain the queries $z_1, \ldots, z_{n-1}$ that are made in this computation. (We do not make these queries at this point. We have not yet used A in any manner.)

(3) Obtain $\mathrm{C}_{2n}^A(x_1, x_3, x_5, \ldots, x_{2n+1}, z_1, z_2, z_3, \ldots, z_{n-1})$, by making $2n$ parallel queries to A.

(4) (We know the status of both x_1 and x_{2n+1} with respect to membership in A.) If $x_1 \notin A$, then $\mathrm{C}_{2n+1}^A(x_1, \ldots, x_{2n+1}) = 0^{2n+1}$, so output 0^{2n+1} and halt. If $x_{2n+1} \in A$, then $\mathrm{C}_{2n+1}^A(x_1, \ldots, x_{2n+1}) = 1^{2n+1}$, so output 1^{2n+1} and halt. If $x_1 \in A$ and $x_{2n+1} \notin A$, go to the next step.

(5) There is a unique $i < n$ such that $\{x_1, x_2, x_3, \ldots, x_{2i+1}\} \subseteq A$ and $\{x_{2i+3}, x_{2i+4}, x_{2i+5}, \ldots, x_{2n+1}\} \subseteq \overline{A}$. We do not yet know whether $x_{2i+2} \in A$, but we do know that

$$\mathrm{C}^A_{2n+1}(x_1, x_2, x_3, \ldots, x_{2n+1}) = \begin{cases} 1^{2i+1}0^{2n+1-(2i+1)}, & \text{if } x_{2i+2} \notin A; \\ 1^{2i+2}0^{2n+1-(2i+2)}, & \text{if } x_{2i+2} \in A. \end{cases}$$

Moreover, $\mathrm{C}^A_{n+1}(x_1, x_3, x_5, \ldots, x_{2n+1}) = 1^{i+1}0^{(n+1)-(i+1)}$, so

$$\mathrm{C}^A_n(x_2, x_4, x_6, \ldots, x_{2n}) = \begin{cases} 1^i0^{n-i}, & \text{if } x_{2i+2} \notin A; \\ 1^{i+1}0^{n-(i+1)}, & \text{if } x_{2i+2} \in A. \end{cases}$$

Hence $x_{2i+2} \in A$ iff $\mathrm{ODD}^A_n(x_2, x_4, x_6, \ldots, x_{2n})$ and i are of opposite parity.

Using the value of $\mathrm{C}^A_{n-1}(z_1, \ldots, z_{n-1})$ from step 3, compute

$b = M^A(x_2, x_4, x_6, \ldots, x_{2n}) = \mathrm{ODD}^A_n(x_2, x_4, x_6, \ldots, x_{2n})$.

Using i and b, compute and output $\mathrm{C}^A_{2n+1}(x_1, \ldots, x_{2n+1})$: There are two cases.

- $b = i \bmod 2$: Then $x_{2i+2} \notin A$, so output $1^{2i+1}0^{2n+1-(2i+1)}$.
- $b \neq i \bmod 2$: Then $x_{2i+2} \in A$, so output $1^{2i+2}0^{2n+1-(2i+2)}$.

∎

4.2.2 ODD^A_n for C.E. Sets A. Our plan is to make c.e. sets *look like* selective sets and then apply a version of Theorem 4.2.

Lemma 4.3. *Let A be a c.e. set, and let X be an extensive set. Then A is selective in X.*

Proof: Choose a computable enumeration $\{A_s\}_{s\in\mathsf{N}}$ of A. Let g be the 0,1-valued computable partial function such that $\mathrm{dom}(g) \subseteq \mathsf{N}^2$ and, for all x, y,

$$g(x,y) = \begin{cases} 1, & \text{if } (\exists s)[x \in A_s \wedge y \notin A_s]; \\ 0, & \text{if } (\exists s)[y \in A_s \wedge x \notin A_s]; \\ \uparrow, & \text{otherwise.} \end{cases}$$

Since X is extensive, there is a 0,1-valued total function $h \leq_{\mathrm{T}} X$ such that h extends g. Now define $f\colon \mathsf{N}^2 \to \mathsf{N}$ by

$$f(x,y) = \begin{cases} x, & \text{if } h(x,y) = 1; \\ y, & \text{if } h(x,y) = 0. \end{cases}$$

It is easy to show that A is selective in X via f. ∎

Theorem 4.4. *Let $n \geq 1$, and let A be c.e. If $\mathrm{ODD}_n^A \in \mathrm{Q}_{||}(n-1, A)$, then A is computable.*

Proof: Suppose that $\mathrm{ODD}_n^A \in \mathrm{Q}_{||}(n-1, A)$. Let X be *any* extensive set (see Definition 1.16). Trivially, $\mathrm{ODD}_n^A \in \mathrm{Q}_{||}^X(n-1, A)$. By Lemma 4.3, A is selective in X. By a relativized version of Theorem 4.2, we have $A \leq_{\mathrm{T}} X$. Since X was *any* extensive set, we have that, for *every* extensive set X, $A \leq_{\mathrm{T}} X$. Since there exist minimal pairs of extensive sets (Theorem 1.18), we obtain that A is computable. ∎

5. Do More Queries Help?

In prior sections, we asked 'How many queries does it take to compute BLAH?' We now ask a more abstract question: 'If I have k queries, can I compute more functions than I could if I had only $k-1$ queries?' The answer depends on what you want to compute (functions or sets) and what you are querying. The short answer is that for computing functions, it always helps to have more queries, but for deciding sets there are cases where more queries do not help.

5.1 More Queries Do Help Compute More Functions!

The results in this section are due to Beigel [Be88]. They later appeared in [GM99].

Theorem 5.1. *If $(\exists n)[\mathrm{FQ}(n, A) = \mathrm{FQ}(n+1, A)]$, then A is computable.*

Proof: By way of contradiction, suppose A is noncomputable. We exhibit a function in $\mathrm{FQ}(n+1, A) - \mathrm{FQ}(n, A)$. If $n = 0$, then $\mathrm{C}_1^A \in \mathrm{FQ}(n+1, A) - \mathrm{FQ}(n, A)$. So assume that $n \geq 1$.

By Corollary 2.3, $\mathrm{C}_{2^n}^A \notin \mathrm{FQ}(n, A)$. Since $\mathrm{C}_n^A \in \mathrm{FQ}(n, A)$, there is some $i \geq n+1$ such that $\mathrm{C}_i^A \notin \mathrm{FQ}(n, A)$ and $\mathrm{C}_{i-1}^A \in \mathrm{FQ}(n, A)$. We show that $\mathrm{C}_i^A \in \mathrm{FQ}(n+1, A)$.

Choose an oracle Turing machine $M^{()}$ so that $\mathrm{C}_{i-1}^A \in \mathrm{FQ}(n, A)$ via M^A. The following algorithm computes C_i^A with at most $n+1$ queries to A. Thus $\mathrm{C}_i^A \in \mathrm{FQ}(n+1, A) - \mathrm{FQ}(n, A)$.

(1) Input $(x_1, \ldots, x_i)$.

(2) Run $M^A(x_1, \ldots, x_{i-1})$. (By hypothesis, M^A makes at most n queries to A.)

(3) Make one additional query to A, namely, "$x_i \in A$?".

(4) Output $\mathrm{C}_i^A(x_1, \ldots, x_i)$ (by concatenating the results of steps 2 and 3).

The theorem is proved. ∎

The next theorem is an analogue of Theorem 5.1 for the bounded-query classes $\mathrm{FQC}(n, A)$, $\mathrm{FQ}_{||}(n, A)$, and $\mathrm{FQC}_{||}(n, A)$. We leave the proof to the reader.

Theorem 5.2. *Let A be a set.*

(1) If $(\exists n)[\mathrm{FQC}(n, A) = \mathrm{FQC}(n+1, A)]$, then A is computable.

(2) If $(\exists n)[\mathrm{FQ}_{||}(n, A) = \mathrm{FQ}_{||}(n+1, A)]$, then A is computable.

(3) If $(\exists n)[\mathrm{FQC}_{||}(n, A) = \mathrm{FQC}_{||}(n+1, A)]$, then A is computable.

5.2 More Queries Do Not Always Help Decide More Sets!

Definition 5.3. $\emptyset^{(\omega)} = \{\langle x, i\rangle \mid x \in \emptyset^{(i)}\}$.

The next theorem shows that when deciding sets by making *parallel* queries to $\emptyset^{(\omega)}$, more queries do not help. We then show that when deciding sets by making *serial* queries to $\emptyset^{(\omega)}$, allowing more queries does enable us to decide more sets. Finally, we exhibit an (unnatural) set such that allowing a greater number of serial queries (with this set as oracle) does not help.

Most of the results in this chapter are in [GM99] but were known many years earlier. They are due to Beigel and Gasarch (no reference available, but I was there). The one exception is Theorem 5.5, which Frank Stephan proved recently and appears here for the first time.

Theorem 5.4. *For all $n \geq 1$, $\mathrm{Q}_{||}(n, \emptyset^{(\omega)}) = \mathrm{Q}(1, \emptyset^{(\omega)})$.*

Proof: Let $n \geq 1$, and let $A \in \mathrm{Q}_{||}(n, \emptyset^{(\omega)})$ via $M^{()}$. Here is an algorithm for $A \in \mathrm{Q}(1, \emptyset^{(\omega)})$.

(1) Input x.

(2) Run $M^{()}(x)$ until the questions $(q_1 \in \emptyset^{(\omega)}, \ldots, q_n \in \emptyset^{(\omega)})$ are asked, but do not try to answer them. Note that, for each i, there exist y_i, z_i such that the question "$q_i \in \emptyset^{(\omega)}$?" is actually the question "$y_i \in \emptyset^{(z_i)}$?" Let z be the max of the z_i.

(3) The question "Is there a set of answers for $y_1 \in \emptyset^{(z_1)}, \ldots, y_n \in \emptyset^{(z_n)}$ that are true and lead to a path of the $M^{()}(x)$ computation that converges to 1?" (note that the answer to this question is yes iff $x \in A$) can be phrased as a query to $\emptyset^{(\omega)}$ (via a query to $\emptyset^{(z+1)}$). Let "$q \in \emptyset^{(\omega)}$?" be that query, and ask it.

(4) If $q \in \emptyset^{(\omega)}$, output 1; otherwise, output 0.

∎

Theorem 5.4 uses the set $\emptyset^{(\omega)}$, which is somewhat natural. Can we use the same set as an example of an oracle for which additional serial queries do not help? As the next result shows, the answer to this question is no.

Theorem 5.5. $\mathrm{Q}(2, \emptyset^{(\omega)}) - \mathrm{Q}(1, \emptyset^{(\omega)}) \neq \emptyset$.

Proof: Let C be the set of ordered pairs (x, y) such that

- $x \in K$, and
- if s is the length of the longest string of 1's on the tape after $M_x(x)$ halts then $y \in \emptyset^{(s)}$. (We assume $s \geq 2$.)

Note that we think of x as an index of a Turing machine and y as an index of an oracle Turing machine.

Clearly, $C \in \mathrm{Q}(2, \emptyset^{(\omega)})$. We show that $C \notin \mathrm{Q}(1, \emptyset^{(\omega)})$. Assume, by way of contradiction, that $C \in \mathrm{Q}(1, \emptyset^{(\omega)})$ via $M^{()}$.

We create Turing machine M_x and oracle Turing machine $M_y^{()}$ such that $M^{\emptyset^{(\omega)}}(x, y) \neq C(x, y)$. The construction of these two machines uses the recursion theorem implicitly.

PROGRAM FOR M_x

(1) Simulate $M^{()}(x, y)$ in such a way that you never write two consecutive 1's on the tape. (E.g., use 00 for 0 and 01 for 1.) Stop the simluation when the one query is made. Let this query be "$q \in \emptyset^{(k)}$?" (Do not make the query.)

(2) Print 01^k0 and then halt. (Hence the longest string of 1's on the tape has length k.)

Note 5.6. Since $x \in K$ and prints out a sequence of k 1's we know that $(x, y) \in C$ iff $y \in \emptyset^{(k)}$.

PROGRAM FOR $M_y^{()}$

(1) Simulate $M^{\emptyset^{(\omega)}}(x, y)$. When the query, "$q \in \emptyset^{(k)}$?", is encountered, make the query. (We will be supplying y with an oracle for $\emptyset^{(k)}$ so this can be done.)

(2) If the simulation outputs 0 (so $M^{\emptyset^{(\omega)}}(x, y)$ thinks that $(x, y) \notin C$ which is equivalent to $y \notin \emptyset^{(k)}$) then halt (which causes $y \in \emptyset^{(k)}$). If the simulation outputs 1 (so $M^{\emptyset^{(\omega)}}(x, y)$ thinks that $(x, y) \in C$ which is equivalent to $y \in \emptyset^{(k)}$) then diverge (which causes $y \notin \emptyset^{(k)}$). The simulation must output something since $M^{\emptyset^{(\omega)}}(x, y)$ computes C.

The key point is that oracle program x is constructed to print out a long enough string of 1's so that oracle program y is able to simulate $M^{\emptyset^{(\omega)}}(x,y)$, make an appropriate query, and diagonalize. It is easy to see that $M^{\emptyset^{(\omega)}}(x,y) \neq C(x,y)$. ∎

The next theorem shows that when deciding sets with sequential queries, more queries do not always help. The set being queried is much less natural than the set used in the previous theorem.

Definition 5.7. A^{tt} is the set of (codes of) Boolean combinations of formulas of the form "v_i" that are true when you interpret v_i to be $i \in A$. For example, the (code of the) formula $(v_{12} \wedge (v_{14} \vee \neg\, v_9))$ is in A^{tt} iff $12 \in A$ and ($14 \in A$ or $9 \notin A$).

Theorem 5.8. *If B is a c.b. set (see Definition 1.19) and $n \geq 1$, then* $\mathrm{Q}(n, B^{\mathrm{tt}}) = \mathrm{Q}(1, B^{\mathrm{tt}})$. *In fact,* $\mathrm{Q}(n, B^{\mathrm{tt}}) = \mathrm{QC}(1, B^{\mathrm{tt}})$.

Proof: Let $n \geq 1$, and let $A \in \mathrm{Q}(n, B^{\mathrm{tt}})$. Clearly, $A \in \mathrm{Q}(n, B^{\mathrm{tt}}) \Rightarrow A \leq_{\mathrm{T}} B$. By Lemma 1.20, $A \leq_{\mathrm{T}} B \Rightarrow A \leq_{\mathrm{tt}} B$. By the definition of $\leq_{\mathrm{tt}}$ and B^{tt}, we easily have that $A \leq_m B^{\mathrm{tt}}$. Clearly, $A \in \mathrm{Q}(1, B^{\mathrm{tt}})$. Note that we actually have $A \in \mathrm{QC}(1, B^{\mathrm{tt}})$. ∎

6. Does Allowing Divergence Help?

The algorithm in the introduction showed that $\mathrm{C}^K_{2^n-1} \in \mathrm{FQ}(n, K)$. For that algorithm, incorrect answers could cause divergence. Is there an algorithm where all query paths converge? More formally, can we obtain $\mathrm{C}^K_{2^n-1} \in \mathrm{FQC}(n, K)$? Also, can we obtain $\mathrm{C}^K_{2^n-1} \in \mathrm{SEN}(2^n)$? Combining the first theorem in this section (which was first proved in [BGGO93]) with Theorem 1.11, we find that the answer to both of these questions is NO—in a strong way.

However, the more general question arises as to when does allowing divergence help. We would like to know whether there are sets A such that any computation with A as an oracle can be replaced with one where all query paths converge. To accomplish this, we explore the question of whether, and to what extent, it helps to allow divergence when incorrect answers are given to one or more of the queries.

The results in this section were stated in [Be*etal* 96]. Full proofs appeared in [GM99].

6.1 A Natural Example of a Function Where Allowing Divergence Helps

We address the question that motivated the study of divergence: is there an algorithm for C_n^K which has all paths converging which asks less than n queries?

Theorem 6.1. *For all $n \geq 1$, $\mathrm{C}_n^K \notin \mathrm{SEN}(2^n - 1)$. (Hence, by Theorem 1.10, $(\forall X)[\mathrm{C}_n^K \notin \mathrm{FQC}(n-1, X)]$.)*

Proof: We offer two proofs. The first one uses the Recursion Theorem. The second one avoids using the Recursion Theorem. We leave as an open problem the question of which proof is better.

A Proof That Uses the Recursion Theorem

Let $n \geq 1$, and suppose, by way of contradiction, that $\mathrm{C}_n^K \in \mathrm{SEN}(2^n - 1)$. Choose a computable function f such that $\mathrm{C}_n^K(x_1, \ldots, x_n) \in D_{f(x_1,\ldots,x_n)}$ and $|D_{f(x_1,\ldots,x_n)}| = 2^n - 1$. By an implicit use of the recursion theorem, we construct programs $a_1, \ldots, a_n$ such that $\mathrm{C}_n^K(a_1, \ldots, a_n) \notin D_{f(a_1,\ldots,a_n)}$.

Program a_i does the following: Compute $f(a_1, \ldots, a_n)$ and determine the set $D_{f(a_1,\ldots,a_n)}$. Find the vector $b_1 b_2 \cdots b_n \notin D_{f(a_1,\ldots,a_n)}$. Halt iff $b_i = 1$.

Programs $a_1, \ldots, a_n$ conspire to make $\mathrm{C}_n^K(a_1, \ldots, a_n) = b_1 b_2 \cdots b_n \notin D_{f(a_1,\ldots,a_n)}$. This is the contradiction.

A Proof That Does Not Use the Recursion Theorem

It is easy to construct a c.e. set A such that, for all n, $\mathrm{C}_n^A \notin \mathrm{SEN}(2^n - 1)$. Since K is m-complete, $A \leq_m K$. One can use this to show that if there is an n such that $\mathrm{C}_n^K \in \mathrm{SEN}(2^n - 1)$ then, for that n, $\mathrm{C}_n^A \in \mathrm{SEN}(2^n - 1)$. Since this is not true, we must have that, for all n, $\mathrm{C}_n^K \notin \mathrm{SEN}(2^n - 1)$. ∎

6.2 A Natural Example of a Set Where Allowing Divergence Does Not Help

Let A, B be sets and $n \in \mathsf{N}$. If $A \in \mathrm{Q}(n, B)$, we can decide whether $x \in A$ by making n queries to B; if the wrong answers are supplied, however, the algorithm may diverge. Is there a set B such that whenever $A \in \mathrm{Q}(n, B)$ we also have $A \in \mathrm{QC}(n, B)$ (that is, even with wrong answers, the algorithm does not diverge)?

By Theorem 5.8, there exists such a set, but it is not natural. We show that K, clearly a natural set, has this property.

Theorem 6.2. *For all $n \in \mathsf{N}$, $\mathrm{Q}(n, K) = \mathrm{QC}(n, K)$.*

Proof: Let $A \in \mathrm{Q}(n, K)$ via M^K. We show that $A \in \mathrm{QC}(n, K)$.

Notation 6.3. We are using $M^{()}$ for our oracle Turing machine. We intend to run it with oracle K. To approximate this we will run it for s steps and use

oracle K_s. This is denoted $M_s^{K_s}$. Do not confuse this with running Turing machine s. The subscript is the number of steps I am running the machine, not an index of a machine.

We first give an intuition behind the proof. Consider the following scenario: Given x, find $M_s^{K_s}(x)$ for $s = 1, 2, 3, \ldots$ until an s_0 is found such that $M_{s_0}^{K_{s_0}}(x) \downarrow = b \in \{0, 1\}$. (Here, the subscripts s, s_0 refer to the number of steps of the computation, not the index of the oracle machine $M^{()}$.) There is no reason to believe that $b = A(x)$; however, we can ask questions about whether at some *later* time the machine (with a better approximation to K) *changes its mind*. These are questions about *mindchanges*. If we find out that the number of mindchanges is even, then b is the answer. If the number of mindchanges is odd, then $1 - b$ is the answer. Note, however, that we never 'run a machine and see what happens' or carry out any other computation that risks diverging.

We now proceed rigorously.

Definition 6.4. Let $M^{()}$ be an oracle Turing machine. Let $x, s_0 \in \mathsf{N}$ and $b \in \{0, 1\}$. Assume that $M_{s_0}^{K_{s_0}}(x) \downarrow = b$. The phrase "there are at least m mindchanges past stage s_0" means that there exist $s_1 < s_2 < s_3 < \cdots < s_m$ such that $s_0 < s_1$, $M_{s_1}^{K_{s_1}}(x) \downarrow = 1 - b$, $M_{s_2}^{K_{s_2}}(x) \downarrow = b$, $M_{s_3}^{K_{s_3}}(x) \downarrow = 1 - b$, $M_{s_4}^{K_{s_4}}(x) \downarrow = b$, etc., and

$$M_{s_m}^{K_{s_m}}(x) \downarrow = \begin{cases} b & \text{if } m \text{ is even;} \\ 1 - b & \text{if } m \text{ is odd.} \end{cases}$$

Note that the question "Are there at least m mindchanges?" can be phrased as a query to K.

Since $M^K(x)$ makes only n queries, there can be at most $2^n - 1$ mindchanges. The following algorithm shows that $A \in \mathrm{QC}(n, K)$.

(1) Input x.

(2) Find the least s_0 such that $M_{s_0}^{K_{s_0}}(x) \downarrow$, and let $b \in \{0, 1\}$ be the output. Note that such an s_0 exists, since $M^K(x) \downarrow$.

(3) Using binary search, one can determine, in n queries to K, how many mindchanges the $M^K(x)$ computation makes past stage s_0. If this number is even, output b; otherwise, output $1 - b$.

Note that the above algorithm converges even if fed the wrong answers. The algorithm never runs any process that might not halt. ∎

6.3 An Unnatural Example of a Set Where Allowing Divergence Helps A Lot

We show that there exists a set A such that $\mathrm{Q}(1,A) - \bigcup_{n=1}^{\infty} \mathrm{QC}(n,A) \neq \emptyset$. In other words, there is a set B that you can decide with just one query to A, provided you allow divergence if wrong answers are given; if you insist that convergence occurs even if one or more of the queries are answered incorrectly, however, then no fixed number of queries suffices.

The results in this chapter are due to Frank Stephan. He never published them; however, they appear in [GM99].

The following easy lemma we leave to the reader.

Lemma 6.5. *For all $A \subseteq \mathsf{N}$ and $n \geq 1$, $\mathrm{QC}(n,A) \subseteq \mathrm{QC}_{||}(2^n-1,A)$. Hence $\bigcup_{n=1}^{\infty} \mathrm{QC}_{||}(n,A) = \bigcup_{n=1}^{\infty} \mathrm{QC}(n,A)$.*

The next lemma restates the problem of getting $B \notin \bigcup_{n=1}^{\infty} \mathrm{QC}(n,A)$ in terms of strong enumerability.

Lemma 6.6. *Let A, B be sets such that $(\exists k \geq 1)[\mathrm{C}_k^A \in \mathrm{SEN}(2^k-1)]$ and $(\forall k \geq 1)[\mathrm{C}_k^B \notin \mathrm{SEN}(2^k-1)]$. Then $B \notin \bigcup_{n=1}^{\infty} \mathrm{QC}(n,A)$. (The intuition behind the statement of this lemma is that A is "easy" and B is "hard," so it is reasonable that B cannot be reduced to A in certain ways.)*

Proof: We show that $B \notin \bigcup_{n=1}^{\infty} \mathrm{QC}_{||}(n,A)$. By Lemma 6.5, we obtain $B \notin \bigcup_{n=1}^{\infty} \mathrm{QC}(n,A)$.

Suppose that $(\exists n_0 \geq 1)[B \in \mathrm{QC}_{||}(n_0,A)]$. By making queries in parallel, we have that $(\forall n \geq 1)[\mathrm{C}_n^B \in \mathrm{FQC}(1, \mathrm{C}_{n_0\, n}^A)]$.

Since $(\exists k \geq 1)[\mathrm{C}_k^A \in \mathrm{SEN}(2^k-1)]$, $\mathrm{C}_{n_0\, n}^A \in \mathrm{SEN}(O(n^k))$ (by Theorem 2.9). Hence $\mathrm{C}_n^B \in \mathrm{SEN}(O(n^k))$. For large enough n, this contradicts the hypothesis on B. ∎

The next definition and lemma restate the problem of getting $B \in \mathrm{Q}(1,A)$ and $(\forall k \geq 1)[\mathrm{C}_k^B \notin \mathrm{SEN}(2^k-1)]$ in terms of fast-growing functions.

Definition 6.7. A function $f : \mathsf{N} \to \mathsf{N}$ is *computably dominated* if there is a computable g such that $(\forall x)[f(x) < g(x)]$.

Lemma 6.8. *Let A be a set such that there exists a function $f \in \mathrm{FQ}(1,A)$ that is not computably dominated. Then there exists $B \in \mathrm{Q}(1,A)$ such that $(\forall k \geq 1)[\mathrm{C}_k^B \notin \mathrm{SEN}(2^k-1)]$.*

Proof: Choose $f \in \mathrm{FQ}(1,A)$ so that f is not computably dominated. We construct a set $B \in \mathrm{Q}(1,A)$ such that, for all $e, k \in \mathsf{N}$ with $k \geq 1$, we satisfy requirement

$$R_{\langle e,k \rangle} : \neg(\mathrm{C}_k^B \text{ is strongly } (2^k-1)\text{-enumerable via } \varphi_e).$$

Let $e, k \in \mathsf{N}$ with $k \geq 1$. If φ_e is not total, requirement $R_{\langle e,k \rangle}$ is automatically satisfied. If φ_e is total and we satisfy $R_{\langle e,k \rangle}$, there will be some k-tuple $(x_1, \ldots, x_k)$ of numbers such that

$$|D_{\varphi_e(x_1,\ldots,x_k)}| \geq 2^k \ \vee \ \mathrm{C}_k^B(x_1, \ldots, x_k) \notin D_{\varphi_e(x_1,\ldots,x_k)}.$$

Choose a computable partition $\{Z_{\langle e,i,k \rangle}\}_{e,i,k \in \mathsf{N}}$ of N so that, for all e, i, k, $|Z_{\langle e,i,k \rangle}| = k$. For all e, i, k with $k \geq 1$, let $\vec{z}_{\langle e,i,k \rangle}$ be the k-tuple of numbers that is formed by taking the elements of $Z_{\langle e,i,k \rangle}$ in increasing numerical order. For all e, k with $k \geq 1$, we intend to satisfy $R_{\langle e,k \rangle}$ by constructing B so that if φ_e computes a total function, then there is some i such that

$$|D_{\varphi_e(\vec{z}_{\langle e,i,k \rangle})}| \geq 2^k \ \vee \ \mathrm{C}_k^B(\vec{z}_{\langle e,i,k \rangle}) \notin D_{\varphi_e(\vec{z}_{\langle e,i,k \rangle})}.$$

We construct $B \in \mathrm{Q}(1, A)$ by giving an algorithm for it.

(1) Input x.

(2) Find e, i, k such that $x \in Z_{\langle e,i,k \rangle}$.

(3) Compute $t = f(i)$. (This requires at most one query to A.)

(4) Compute $M_{e,t}(\vec{z}_{\langle e,i,k \rangle})$.

(5) There are two cases.

   (a) $M_{e,t}(\vec{z}_{\langle e,i,k \rangle}) \uparrow$: Output 0. (We have not made progress towards satisfying requirement $R_{\langle e,k \rangle}$.)

   (b) $M_{e,t}(\vec{z}_{\langle e,i,k \rangle}) \downarrow = y$: There are two cases.

   - $|D_y| \geq 2^k$: Output 0. (Note that requirement $R_{\langle e,k \rangle}$ is automatically satisfied.)
   - $|D_y| \leq 2^k - 1$: We want to set $B(x)$, and for that matter $B(z)$ for all $z \in Z_{\langle e,i,k \rangle}$, such that $\mathrm{C}_k^B(\vec{z}_{\langle e,i,k \rangle}) \notin D_y$. For now, we can set only $B(x)$. Find σ, the lexicographically least string in $\{0,1\}^k - D_y$, and let j be such that x is the j^{th} component of $\vec{z}_{\langle e,i,k \rangle}$. Output $\sigma(j)$. (Note that for all $z \in Z_{\langle e,i,k \rangle}$, running this algorithm on input z will get us to this same step and will yield the same σ; hence we will have $\mathrm{C}_k^B(\vec{z}_{\langle e,i,k \rangle}) = \sigma \notin D_y$, and $R_{\langle e,k \rangle}$ will be satisfied.)

Let $e, k \in \mathsf{N}$ such that $k \geq 1$. We show that $R_{\langle e,k \rangle}$ is satisfied. If φ_e is not total, then clearly $R_{\langle e,k \rangle}$ is satisfied. Assume, by way of contradiction, that φ_e is total and $R_{\langle e,k \rangle}$ is not satisfied. We use this to obtain a computable function g that dominates f, in contradiction to our assumption about f.

Since $R_{\langle e,k\rangle}$ is not satisfied, we know that, for every i, the $M_e(\vec{z}_{\langle e,i,k\rangle})$ computation does not halt within $f(i)$ steps. (Otherwise, $R_{\langle e,k\rangle}$ would have been satisfied when the elements of $Z_{\langle e,i,k\rangle}$ were input to the algorithm.) Since φ_e is total, the following computable function dominates f:

$$g(i) = \mu t[M_{e,t}(\vec{z}_{\langle e,i,k\rangle}) \downarrow].$$

Thus $R_{\langle e,k\rangle}$ *is* satisfied. ∎

To obtain our result from Lemmas 6.6 and 6.8, it suffices to have a set A such that

- $(\exists k \geq 1)[\mathrm{C}_k^A \in \mathrm{SEN}(2^k - 1)]$ and
- there exists $f \in \mathrm{Q}(1, A)$ such that f is not computably dominated.

These two properties seem hard to obtain at the same time, since the first one says that A is "easy" while the second one says that A is "hard." Even so, the following lemma allows us to obtain such sets easily.

Lemma 6.9.

(1) If A is selective, then $(\exists k \geq 1)[\mathrm{C}_k^A \in \mathrm{SEN}(2^k - 1)]$. (Actually, $(\forall k \geq 1)[\mathrm{C}_k^A \in \mathrm{SEN}(k + 1)]$.)

(2) If A is a noncomputable c.e. set, then there is a function $f \in \mathrm{FQ}(1, A)$ such that f is not computably dominated. (This is well known.)

(3) There exist noncomputable c.e. sets that are selective. (This is from [Jo68]. The proof I present uses a set defined by Dekker [De54].)

Proof:

1) This follows from Lemma 1.15.

2) Choose a computable enumeration $\{A_s\}_{s\in\mathsf{N}}$ of A, and let f be the function defined by

$$f(x) = \begin{cases} \mu s[x \in A_s], & \text{if } x \in A; \\ 0, & \text{otherwise.} \end{cases}$$

Clearly, $f \in \mathrm{FQ}(1, A)$. Suppose f is computably dominated, and choose a computable g so that $(\forall x)[f(x) < g(x)]$. Then

$$(\forall x)[x \in A \text{ iff } x \in A_{g(x)}].$$

This demonstrates that A is computable, a contradiction!

3) Let C be a noncomputable c.e. set. Choose a computable enumeration $\{C_s\}_{s\in\mathsf{N}}$ of C such that at every stage exactly one new element comes in. Let c_s be the new element that comes in at stage s, and let

$$A = \{s \mid (\exists t > s)[c_t < c_s]\}.$$

Clearly, A is c.e. Using Definition 1.13.2, it is easy to show A selective. ∎

Theorem 6.10. *There exists A such that* $\mathrm{Q}(1, A) - \bigcup_{n=1}^{\infty} \mathrm{QC}(n, A) \neq \emptyset$.

Proof: This follows from Lemmas 6.6, 6.8, and 6.9. ∎

Note 6.11. The following is known. Let A be a set.

(1) There exists $B \equiv_{\mathrm{tt}} A$ such that $(\forall n)[\mathrm{Q}(n, B) = \mathrm{QC}(n, B)]$ iff all $f \leq_{\mathrm{wtt}} A$ are computably dominated.

(2) There exists $B \equiv_{\mathrm{tt}} A$ such that $\mathrm{Q}(1, B) - \bigcup_{i=1}^{\infty} \mathrm{QC}(n, B) \neq \emptyset$ iff there exists $f \leq_{\mathrm{wtt}} A$ such that f is not computably dominated.

7. Does Order Matter?

Let $A, B \subseteq \mathsf{N}$. I am allowed to make one query to A *and* one query to B in some computation. Does the order in which I make the queries matter? The first theorem in this section is due to Beigel (unpublished) and has been generalized by McNicholl [McN00]. The second theorem is due to McNicholl [McN00]; however, the proof given here is due to Frank Stephan and has not been published previously. Questions of this type were asked in complexity theory [HHW98] before they were asked in computability theory.

Definition 7.1. Let $A, B \subseteq \mathsf{N}$. $\mathrm{QO}(A, B)$ is the set of sets that I can decide by an algorithm that makes one query to A and then one query to B. (The 'O' stands for 'order.') $\mathrm{Q}_{||}(A, B)$ is the set of sets that I can decide by an algorithm that makes one query to A and one query to B at the same time. (This differs from the use of $\mathrm{Q}_{||}(n, A)$ used earlier in this paper.)

Definition 7.2. If $\mathrm{QO}(A, B) = \mathrm{QO}(B, A)$, then A and B *commute*.

Notation 7.3. We denote an oracle Turing machine that is going to query two oracles, with one query each, by $M^{()()}$. We fill in the first $()$ with the oracle it queries first, and the second $()$ with the oracle it queries second. If $b_1b_2 \in \{0,1\}^2$ then $M^{b_1b_2}(x)$ denotes what happens if you assume the answer to the first question is b_1 and the answer to the second question (if such a question exists) is b_2. (Note that you do not actually query either oracle.) If $B \subseteq \mathsf{N}$,

then (1) $M^{(b_1)(B)}(x)$ denotes what happens if you assume the answer to the first question is b_1 but you get the true answer to the second question (if such a question exists) by querying B, and (2) $QUERY(M^{(b_1)(B)}(x))$ denotes the query to B that is encountered in the $M^{(b_1)(B)}(x)$ computation, if it exists (if it does not exist, then $QUERY(M^{(b_1)(B)}(x))$ is undefined). If we use this notation, we are implicitly assuming that the query to B exists.

We leave the proof of the following easy lemma to the reader.

Lemma 7.4. *Let $i, j, q, x \in \mathsf{N}$, $b_1, b_2, b \in \{0, 1\}$, and $M^{()()}$ be an oracle Turing machine. Assume $i < j$. Then the following hold.*

(1) If $i \geq 1$, then the truth value of the statement

$$(q \in \emptyset^{(i)}) \wedge (M^{b_1 b_2}(x) \downarrow = b)$$

can be determined by a query to $\emptyset^{(i)}$.

(2) If $i \geq 2$, then the truth value of the statement

$$(q \notin \emptyset^{(i)}) \wedge (M^{b_1 b_2}(x) \downarrow = b)$$

can be determined by a query to $\emptyset^{(i)}$.

(3) The truth value of the statement

$$(q \in \emptyset^{(j)}) \wedge (M^{(b_1)(\emptyset^{(i)})}(x) \downarrow = b)$$

can be determined by a query to $\emptyset^{(j)}$.

(4) The truth value of the statement

$$(q \notin \emptyset^{(j)}) \wedge (M^{(b_1)(\emptyset^{(i)})}(x) \downarrow = b)$$

can be determined by a query to $\emptyset^{(j)}$.

(5) The truth value of the statement

$$\begin{aligned}&[(q \in \emptyset^{(i)}) \wedge (QUERY(M^{(b_1)(\emptyset^{(j)})}(x)) \in \emptyset^{(j)})]\\ &\vee[(q \notin \emptyset^{(i)}) \wedge (QUERY(M^{(b_2)(\emptyset^{(j)})}(x)) \in \emptyset^{(j)})]\end{aligned}$$

can be determined by a query to $\emptyset^{(j)}$.

Theorem 7.5. *For all $i, j \geq 1$, $\emptyset^{(i)}$ and $\emptyset^{(j)}$ commute. In fact,*

$$\mathrm{QO}(\emptyset^{(i)}, \emptyset^{(j)}) = \mathrm{QO}(\emptyset^{(j)}, \emptyset^{(i)}) = \mathrm{Q}_{||}(\emptyset^{(j)}, \emptyset^{(i)}).$$

Proof: Assume $i < j$. Let $C \in \mathrm{QO}(\emptyset^{(i)}, \emptyset^{(j)})$ via $M^{\emptyset^{(i)},\emptyset^{(j)}}$. We show that $C \in \mathrm{Q}_{||}(\emptyset^{(j)}, \emptyset^{(i)})$. The intuition is that we can ask the question "what is the answer to the second question going to be" and the question "what is the answer to the first question" at the same time.

(1) Input x.

(2) Run the $M^{\emptyset^{(i)},\emptyset^{(j)}}(x)$ computation until the first query is encountered. Call this query q (we do not make this query).

(3) Consider the following statement:

$$[(q \in \emptyset^{(i)}) \wedge (QUERY(M^{(1)(\emptyset^{(j)})}(x)) \in \emptyset^{(j)})]$$
$$\vee[(q \notin \emptyset^{(i)}) \wedge (QUERY(M^{(0)(\emptyset^{(j)})}(x)) \in \emptyset^{(j)})]$$

By Lemma 7.4.5, this can be phrased as a query z to $\emptyset^{(j)}$. note that $z \in \emptyset^{(j)}$ iff the second query of the $M^{\emptyset^{(i)},\emptyset^{(j)}}$ computation has the answer YES.

(4) Ask "$z \in \emptyset^{(j)}$?" and "$q \in \emptyset^{(i)}$?" at the same time. This will give you all the information you need to simulate the computation of $M^{\emptyset^{(i)},\emptyset^{(j)}}(x)$.

Now let $C \in \mathrm{QO}(\emptyset^{(j)}, \emptyset^{(i)})$ via $M^{\emptyset^{(j)},\emptyset^{(i)}}$. We show that $C \in \mathrm{QO}(\emptyset^{(i)}, \emptyset^{(j)})$.

We first prove this for the case where $i \geq 2$. This is needed, since Lemma 7.4.2 does not hold when $i = 1$. We will prove the $i = 1$ case later.

The intuition is that we first find an approximation to the answer by seeing which query path converges first, and then ask about mindchanges. We first ask if there is a mindchange because of the second query, and then we ask if there is a mindchange because of the first query.

(1) Input x.

(2) Run $M^{()()}(x)$ along all query paths until one of them halts. (At least one must halt, since the correct answers yield a halting path.) Let b be the output on the halting path, let q_1 be the first query encountered (the query to $\emptyset^{(j)}$), and let q_2 be the second query encountered (a query to $\emptyset^{(i)}$). We will be asking questions about whether the computation wants to change its mind about b. There are four cases, corresponding to the four possible pairs of answers supplied to the queries.

(3) (a) Case 1: The answers 0,0 yield a halting path. By Lemma 7.4.1, we can determine the truth value of the following statement by making a query to $\emptyset^{(i)}$:

$$(q_2 \in \emptyset^{(i)}) \wedge (M^{01}(x) \downarrow = 1 - b).$$

Make the query. If the answer is YES, let $c = 1 - b$; otherwise, let $c = b$. If $q_1 \notin \emptyset^{(j)}$ (do not ask this), then the final correct answer is c. By Lemma 7.4.3, we can determine the truth value of the following statement by making a query to $\emptyset^{(j)}$:

$$(q_1 \in \emptyset^{(j)}) \wedge (M^{(1)(\emptyset^{(i)})}(x) \downarrow = 1 - c).$$

Make the query. If the answer is YES, output $1 - c$; otherwise, output c.

(b) Case 2: The answers 0,1 yield a halting path. By Lemma 7.4.2 (and $i \geq 2$), we can determine the truth value of the following statement by making a query to $\emptyset^{(i)}$:

$$(q_2 \notin \emptyset^{(i)}) \wedge (M^{00}(x) \downarrow = 1 - b).$$

Make the query. If the answer is YES, let $c = 1 - b$; otherwise, let $c = b$. If $q_1 \notin \emptyset^{(j)}$ (do not ask this), then the final correct answer is c. The rest of this case is identical to Case 1.

(c) Case 3: The answers 1,0 yield a halting path. By Lemma 7.4.1, we can determine the truth value of the following statement by making a query to $\emptyset^{(i)}$:

$$(q_2 \in \emptyset^{(i)}) \wedge (M^{11}(x) \downarrow = 1 - b).$$

Make the query. If the answer is YES, let $c = 1 - b$; otherwise, let $c = b$. If $q_1 \in \emptyset^{(j)}$ (do not ask this), then the final correct answer is c. By Lemma 7.4.4, we can determine the truth value of the following statement by making a query to $\emptyset^{(j)}$:

$$(q_1 \notin \emptyset^{(j)}) \wedge (M^{(0)(\emptyset^{(i)})}(x) \downarrow = 1 - c).$$

Make the query. If the answer is YES, output $1 - c$; otherwise, output c.

(d) Case 4: The answers 1,1 yield a halting path. By Lemma 7.4.2 (and $i \geq 2$), we can determine the truth value of the following statement by making a query to $\emptyset^{(i)}$:

$$(q_2 \notin \emptyset^{(i)}) \wedge (M^{10}(x) \downarrow = 1 - b).$$

Make the query. If the answer is YES, let $c = 1 - b$; otherwise, let $c = b$. If $q_1 \in \emptyset^{(j)}$ (do not ask this), then the final correct answer is c. The rest of the proof is identical to Case 3.

We now look at the case where $i = 1$. We need to show that if $C \in \mathrm{QO}(\emptyset^{(j)}, K)$, then $C \in \mathrm{QO}(K, \emptyset^{(j)})$. Assume $C \in \mathrm{QO}(\emptyset^{(j)}, K)$ via $M^{()()}$. We show that $C \in \mathrm{QO}(\emptyset^{(i)}, \emptyset^{(j)})$.

(1) Input x.

(2) Run all query paths of the $M^{()()}(x)$ machine. If a second query is encountered (a query to K), then before pursuing the YES path, enumerate K and wait for the element to enter. (If it never enters K, there is no point in pursuing the YES path.) Wait until one of the four query paths halts—with the caveat about the existence of a second query and pursuit of the YES path for that query. (At least one must halt, since the correct answers yield a halting path.) Let b be the answer on the halting path, let q_1 be the first query encountered (the query to $\emptyset^{(j)}$), and let q_2 be the second query encountered (a query to K). We will be asking questions about whether the computation wants to change its mind about b. Note that if there is a mindchange because of the first query, then q_2 may change (that is, the actual query made to K may change). If it does, the answer to the new q_2 may differ from the one supplied for q_2 on the original halting query path (even if that answer was verified by enumeration of K). There are four cases, corresponding to the four possible pairs of answers supplied to the queries.

(3) (a) Case 1: The answers 0,0 yield a halting path. This is identical to Case 1 in the previous algorithm.

(b) Case 2: The answers 0,1 yield a halting path. The query to K was answered correctly. Note that if $q_1 \notin \emptyset^{(j)}$ (do not ask this), then the final correct answer is b. The rest of this case is identical to Case 1.

(c) Case 3: The answers 1,0 yield a halting path. This is identical to Case 3 in the previous algorithm.

(d) Case 4: The answers 1,1 yield a halting path. The query to K was answered correctly. Note that if $q_1 \in \emptyset^{(j)}$ (do not ask this), then the final correct answer is b. The rest of the proof is identical to Case 4 in the previous algorithm.

∎

We show that, for all i, $\emptyset^{(i)}$ and $\emptyset^{(\omega)}$ do not commute. We first need a lemma of interest in its own right.

Lemma 7.6. *For all i,* $\mathrm{QO}(\emptyset^{(\omega)}, \emptyset^{(i)}) \subseteq \mathrm{Q}(1, \emptyset^{\omega})$.

Proof: Let $A \in \mathrm{QO}(\emptyset^{(\omega)}, \emptyset^{(i)})$ via $M^{\emptyset^{\omega}, \emptyset^{(i)}}$. The following algorithm shows $A \in \mathrm{Q}(1, \emptyset^{\omega})$. The intuition is that once we know the first query we can ask a complex query about answering the first one and the rest of the computation.

(1) Input x

(2) Run $M^{()()}(x)$ until a query $q \in \emptyset^\omega$ is encountered. Do not ask this query.

(3) Find z, k such that the query is actually of the form $z \in \emptyset^{(k)}$.

(4) Phrase the query

$$[(z \in \emptyset^{(k)}) \wedge (M^{(1)(\emptyset^{(i)})}(x)) \downarrow = 1]$$
$$\vee[(z \notin \emptyset^{(k)}) \wedge (M^{(0)(\emptyset^{(i)})}(x)) \downarrow = 1]$$

as a query y to $\emptyset^\omega$.

(5) Ask $y \in \emptyset^\omega$. If $y \in \emptyset^\omega$ then output YES, else output NO.

The lemma follows. ∎

Theorem 7.7. *K and $\emptyset^{(\omega)}$ do not commute.*

Proof: Let C be the set from Theorem 5.5. Clearly, $C \in \mathrm{QO}(\emptyset^{(i)}, \emptyset^{(\omega)})$. We show that $C \notin \mathrm{QO}(\emptyset^{(\omega)}, \emptyset^{(i)})$. Assume, by way of contradiction, that $C \in \mathrm{QO}(\emptyset^{(\omega)}, \emptyset^{(i)})$. By Lemma 7.6 we have $C \in \mathrm{Q}(1, \emptyset^{(\omega)})$. By Theorem 5.5 $C \notin \mathrm{Q}(1, \emptyset^{(\omega)})$. Hence we have our contradiction. ∎

8. Acknowledgments

I would like to thank Richard Beigel, Fawzi Emad, Lance Fortnow, Omer Horovitz, Georgia Martin, Jim Owings, and Frank Stephan for proofreading and commentary. I particularly want to thank Georgia Martin for her meticulous proofreading and Frank Stephan for some new proofs. In addition I would like to thank the referee, who has likely already been thanked, for helpful suggestions.

References

[Be87] Beigel, Richard (1987). *Query-Limited Reducibilities*. PhD thesis, Stanford University. Also available as Report No. STAN-CS-88–1221.

[Be88] Beigel, Richard (1988). When are $k+1$ queries better than k? Technical Report 88-06, The Johns Hopkins University, Dept. of Computer Science.

[BGGO93] Beigel, Richard, Gasarch, William, Gill, John, and Owings, James (1993). Terse, superterse, and verbose sets. *Information and Computation*, 103(1):68–85.

[Be*etal* 00] Beigel, Richard, Gasarch, William, Kummer, Martin, Martin, Georgia, McNicholl, Timothy, and Stephan, Frank (2000). The complexity of ODD_n^A. *Journal of Symbolic Logic*, pages 1–18.

[Be*etal* 96] Beigel, Richard, Gasarch, William, Kummer, Martin, Martin, Georgia, McNicholl, Timothy, and Stephan, Frank (1996). On the query complexity of sets. In *21st International Symposium on Mathematical Foundations of Computer Science (MFCS '96), Cracow, Poland*.

[BEHW89] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36:929–965.

[CH89] Cai, Jin-yi and Hemachandra, Lane A. (1989). Enumerative counting is hard. *Information and Computation*. Earlier verison in Structures 1988.

[COS75] Clarke, Steven, Owings, Jim, and Spriggs, James (1975). Trees with full subtrees. In *Proc. of the 6th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 169–172.

[De54] Dekker, J. C. E. (1954). A theorem on hypersimple sets. *Proceedings of the AMS*, 5:791–796.

[Ga85] Gasarch, William (1985). A hierarchy of functions with applications to recursive graph theory. Technical Report 1651, University of Maryland, Dept. of Computer Science.

[GM99] Gasarch, William and Martin, Georgia (1999). *Bounded Queries in Recursion Theory*. Progress in Computer Science and Applied Logic. Birkhäuser, Boston.

[HW87] Haussler, D. and Welzl, E. (1987). ϵ-nets and simplex range queries. *Discrete Computational Geometry*, 2:127–151.

[HHW98] Hemaspaandra, Hempel, and Wechsung (1998). Query order. *sicomp*, 28.

[Jo68] Jockusch, Carl (1968). Semirecursive sets and positive reducibility. *Transactions of the AMS*, 131:420–436.

[Jo89] Jockusch, Carl (1989). Degrees of functions with no fixed points. In Fenstad, J.E., Frolov, I., and Hilpinen, R., editors, *Logic, Methodology, and Philosophy of Science VIII*, pages 191–201. North Holland.

[JS72] Jockusch, Carl and Soare, Robert (1972). Π^0_1 classes and degrees of theories. *Transactions of the AMS*, 173:33–56.

[Ku92] Kummer, Martin (1992). A proof of Beigel's cardinality conjecture. *Journal of Symbolic Logic*, 57(2):677–681.

[KS94] Kummer, Martin and Stephan, Frank (1994). Effective search problems. *Mathematical Logic Quarterly*, 40:224–236.

[McN00] McNichol, Tim (2000). On the commutativity of jumps. *Journal of Symbolic Logic*, 65(4).

[MM68] Miller, Webb and Martin, Donald A. (1968). The degree of hyperimmune sets. *Zeitsch. f. math. Logik und Grundlagen d. Math.*, 14:159–166.

[Od89] Odifreddi, Piergiorgio (1989). *Classical Recursion Theory (Volume I)*. North-Holland, Amsterdam.

[Ow89] Owings, Jr., James C. (1989). A cardinality version of Beigel's Nonspeedup Theorem. *Journal of Symbolic Logic*, 54(3):761–767.

[Sa72] Sauer, N. (1972). On the density of families of sets. *Journal of Combinatorial Theory (series A)*, 13:145–147.

[Sc62] Scott, Dana (1962). Algebras of sets binumerable in complete extension of arithmetic. In *Proceedings Symposium Pure and Applied Mathematics 5*, pages 117–121.

[Sh72] Shelah, S. (1972). A combinatorial problem: stability and order for models and theories of infinitary languages. *Pacific Journal of Mathematics*, 41:247–261.

[So87] Soare, Robert (1987). *Recursively Enumerable Sets and Degrees*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin.

[So96] Soare, Robert (1996). Computability and recursion. *Bulletin of Symbolic Logic*, 27.

[VC71] Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probab. and its Applications*, 16(2):264–280.