QUERY-LIMITED REDUCIBILITIES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Richard Beigel
January 1995

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

John T. Gill, III
Electrical Engineering
(Principal Adviser)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Robert W Floyd

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Vaughan R. Pratt

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies & Research

Abstract

We study classes of sets and functions computable by algorithms that make a limited number of queries to an oracle. We distinguish between queries made in parallel (each question being independent of the answers to the others, as in a truth-table reduction) and queries made in serial (each question being permitted to depend on the answers to the previous questions, as in a Turing reduction).

We define computability by a set of functions, and we show that it captures the information-theoretic aspects of computability by a fixed number of queries to an oracle. Using that concept, we prove a very powerful result, the Nonspeedup Theorem, which states that 2^n parallel queries to any fixed nonrecursive oracle cannot be answered by an algorithm that makes only n queries to any oracle whatsoever. This is the tightest general result possible. A corollary is the intuitively obvious, but nontrivial result that additional parallel queries to an oracle allow us to compute additional functions; the same is true of serial queries.

We show that if $k + 1$ parallel queries to the oracle A can be answered by an algorithm that makes only k serial queries to any oracle B, then n parallel queries to the oracle A can be answered by an algorithm that makes only $O(\log n)$ parallel queries to a third oracle C.

We also consider polynomial time bounded algorithms that make a fixed number of queries to an oracle. It has been shown that the Nonspeedup Theorem does not apply in the polynomial time bounded framework. However, we prove a Weak Nonspeedup Theorem, which states that if 2^k parallel queries to the oracle A can be answered by an algorithm that makes only k serial queries to the oracle B, then any n parallel queries to the oracle A can be answered by an algorithm that makes only $2^k - 1$ *of*

the same queries to A. A corollary is that if A is NP-hard and $\mathsf{P} \neq \mathsf{NP}$, then extra parallel queries to A allow us to compute extra functions in polynomial time; the same is true of serial queries.

Acknowledgements

Many thanks are due to Jon Siegel for accidentally inspiring the topic of bounded query classes. I would like to thank Al Aho and Don Knuth for commenting on early versions of this work; I especially thank Don Knuth for his early encouragement of this research and for his support while I was looking for an advisor.

I warmly thank my readers, Vaughan Pratt and Bob Floyd. I especially thank Bob Floyd for encouraging me throughout the course of this research, for repeatedly telling me to generalize, for helping me to find an adviser, for simplifying and illuminating many of my proofs, and for being a personal friend. I am grateful to my advisor, John Gill, for his endless patience and for his repeated requests that I prove things that I thought I could not prove.

I thank my regular collaborators, Amihood Amir, Bill Gasarch, Louise Hay, and Jim Owings, whose contributions to notation and whose comments on our co-authored articles have added much to the clarity of exposition in this dissertation. In addition, I thank Art Delcher, Simon Kasif, Cathy Schevon, and Dwight Wilson, whose comments on various technical reports and articles have also improved this exposition.

Thanks are due to Greg Sullivan and Mike Goodrich for proofreading the introduction and the conclusions of this dissertation. Special thanks are due to Bill Gasarch and Jim Owings for allowing me to include excerpts from their work, and for proofreading the final version of this dissertation.

This dissertation was typeset using Leslie Lamport's LaTeX macro package for Donald E. Knuth's TeX typesetting system. Early versions of this dissertation were prepared at Stanford University on a machine named Sail, which is run in exemplary fashion by Martin Frost; money for computer time was provided by the Department of Computer Science. The final version of this dissertation was prepared at The Johns Hopkins University, with money for computer time provided by the Dean of Engineering.

This research was supported by fellowships from the National Science Foundation and from the Fannie and John Hertz Foundation.

Contents

Abstract iii

Acknowledgements v

1 Introduction 1

2 Preliminaries 4

2.1 Terminology and Conventions 4

2.2 Observations about Q and FQ 10

3 Bounded Queries to the Halting Problem 22

3.1 Lemmas About K 22

3.2 Separating the Bounded Query Classes 26

3.3 A Normal Form for Languages in $\mathrm{Q}_{||}(n, K)$ 33

3.4 Several Rounds of Parallel Queries 36

3.5 The Query Complexity Measure 38

3.5.1 Halting problems for K-machines 40

3.5.2 Recursively Defined Halting Problems 45

3.6 Unbounded Queries 49

3.7 Chromatic Number of a Recursive Graph 56

3.8 Related Work 59

4 Nonrecursive Oracles 61

4.1 Computability by a Set of Partial Recursive Functions 61

4.2 The Nonspeedup Theorem 64
4.3 Separation Theorems 67
4.4 Decision Problems 69
4.5 Terse and Superterse Sets 74
4.6 Verbose Sets 76
4.7 $\#_{2^n}^A \in ?\ \mathrm{FQ}(n, B)$ 78
4.8 Quantifying Verboseness 81
4.9 Decision Problems and Superterseness 91
4.10 Discussion and Related Work 93

5 Polynomial Time Bounded Reductions **95**
5.1 Computability by a Set of Polynomial Time Functions 96
5.2 A Weak Nonspeedup Theorem 97
5.3 A Serial-Parallel Tradeoff 102
5.4 Cheatable Sets 103
5.5 P-Terse and P-Verbose Sets 111
5.6 Decision Problems and P-terse Sets 114
5.7 NP-Hard and Δ_2^{P}-Hard Oracles 115
5.8 Related Work 120

6 Conclusions **122**

A Chromatic Number of a Recursive Graph **127**

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation is concerned entirely with computations that make use of an oracle. An oracle is an imaginary device with which we equip an ordinary computer, in order to give the computer additional computational power. Each oracle is associated with a particular set A of strings (or natural numbers); the oracle is said to be an *oracle for the set A*. When the computer needs to know if the string (or natural number) x belongs to A, the computer asks the oracle; the computer is then able to use the oracle's answer in the remainder of the computation. Although this dissertation will not be concerned with machine model issues, we refer to Hopcroft and Ullman's model of oracle computation on a Turing machine for the sake of completeness of exposition [HU79, Section 8.9, pp. 209-210]:

> Let A be a language, $A \subseteq \Sigma^*$. A *Turing machine with oracle A* is a single-tape Turing machine with three special states $q_?$, q_y, and q_n. The state $q_?$ is used to ask whether a string is in the set A. When the Turing machine enters the state $q_?$ it requests an answer to the question: "Is the string of nonblank symbols to the right of the tape head in A?" The answer is supplied by having the state of the Turing machine change on the next move to one of the two states q_y or q_n, depending on whether the answer is yes or no. The computation continues normally until the next time $q_?$ is entered, when the "oracle" answers another question.

In this way, an oracle allows us to consider the questions, "What if a computer could solve ⟨your favorite unsolvable problem⟩?" and "What if a computer could solve ⟨your favorite intractable problem⟩ efficiently?" without being faced with an immediate logical contradiction.

Oracle computations allow us to formalize the notion that one problem is computationally more difficult than another. In much previous work [Rog67, HU79, Soa87], the set B has been said to be more computationally difficult than the set A if A is decidable by a computer with an oracle for B, but B is not decidable by any computer with an oracle for A. That notion is essentially qualitative.

In this dissertation, we adopt a quantitative notion of when one problem is computationally more difficult than another. We fix a set C and we assume that our computers are equipped with an oracle for C. We say that the set B is computationally more difficult than the set A if A is decidable by a computer that makes only k queries to the oracle for C for some constant k, but B is not decidable by any computer that makes only k queries to the oracle for C.

Our notion of computational difficulty gives rise to a natural complexity measure,[1] the query complexity for oracle computations. The query complexity of a computation is the number of queries that the computation makes to its oracle. In Section 2.1, we define the bounded query classes, which are classes of languages decidable by a computer that makes a fixed number of queries to a fixed oracle. The bounded query classes are complexity classes of the query complexity measure.

Throughout this dissertation we examine the following question: "When does the ability to ask $n+1$ queries to an oracle for A allow us to solve harder problems than we could solve with only n queries?" This general question admits several variants, depending on the following issues:

- What do we mean by asking n or $n+1$ queries? Must all queries be made in parallel (each question being independent of the answers to the others, as in a truth-table reduction), or may the queries be made in series (each question

[1]This measure is not always a computational complexity measure in the sense of Blum (see Section 3.5).

being permitted to depend on the answers to the previous questions, as in a Turing reduction)?

- What do we mean by problems? Are we considering the difficulty of computing functions, or of solving decision problems?
- What restrictions do we place on the oracle? Must the n queries be posed to the oracle for A, as are the $n+1$ queries, or may they be posed to a different oracle?

In Chapter 3 we study the bounded query classes relative to an oracle for the halting problem, and we prove a variety of separation results. In particular, $n+1$ queries to an oracle for the halting problem allow us to solve more decision problems than we can solve by making only n queries to an oracle for the halting problem, as long as we are consistent about serial and parallel queries.

In Chapter 4 we study the bounded query classes relative to an oracle for an arbitrary nonrecursive set, and we generalize some of the results from Chapter 3. We prove a very powerful result, the Nonspeedup Theorem, which says that 2^n parallel queries to a nonrecursive oracle cannot be answered by an algorithm that makes only n queries to any oracle whatsoever. This is the tightest general result possible. One of its corollaries is that $n+1$ queries to an oracle for the nonrecursive set A allow us to compute more functions than n queries to an oracle for the same set A allow us to compute, as long as we are consistent about serial and parallel queries.

In Chapter 5 we study bounded query classes within a polynomial time bounded setting. Unfortunately, the Nonspeedup Theorem does not generalize in the way we would first expect. In fact, Amir and Gasarch have constructed a set $A \notin \mathsf{P}$ such that n queries to an oracle for A *do not* allow us to compute more functions than we can compute by making a single query to an oracle for A [AG88]. The techniques used in Chapter 5 are therefore more subtle and oracle-specific than those of Chapter 4. We show that if A is NP-hard and $\mathsf{P} \neq \mathsf{NP}$ then $n+1$ queries to an oracle for the set A allow us to compute in polynomial time more functions than we can compute in polynomial time by making only n queries to an oracle for the same set A, as long as we are consistent about serial and parallel queries.

Chapter 2

Preliminaries

2.1 Terminology and Conventions

> *When* I *use a word, it means just what I choose it to mean — neither more nor less.*
>
> — Humpty Dumpty [Car62]

We write $A \subseteq B$ to denote that A is a subset of B; $A \subset B$ to denote that A is a proper subset of B; $\bar{A}$ to denote the complement[1] of the set A; $A - B$ to denote the set difference $A \cap \bar{B}$; $\max A$ to denote the maximum element of the finite set A, 0 if A is empty; $|A|$ or $\mathrm{card}(A)$ to denote the cardinality of the set A; and $\chi_A(x)$ to denote the characteristic function of the set A: 1 if $x \in A$, 0 if $x \notin A$. We write N to denote the set of natural numbers. We write $p \vee q$ to denote the the inclusive-or of the two logical values p and q: 1 if $p = 1$ or $q = 1$, 0 otherwise; and we write $p \oplus q$ to denote the exclusive-or of p and q: 1 if $p \neq q$, 0 otherwise. We always use base-2 logarithms.

We assume that the reader has a basic familiarity with recursion theory, including Turing machines; partial recursive and total recursive functions; recursive and recursively enumerable sets; and many-one and Turing reductions. These concepts

[1] Our default universal set is the set of all strings over some fixed alphabet. Since there is an effective (polynomial time computable, in fact) one-one correspondence between the set of all strings over a fixed alphabet and the set of natural numbers, we can just as easily take our default universal set to be the set of natural numbers.

are explained in [Rog67, HU79, Soa87]. The footnotes occasionally refer to more advanced material in recursion theory that is not necessary in order to understand the results in this dissertation.

Because we are not concerned with the particulars of our machine model, we will use the following terms synonymously: computer, Turing machine, program, algorithm, machine. Using standard dovetailing techniques [HU79, Section 7.7], we can run countably many computations at once. Thus, we can construct machines that "timeshare" several computations or run several computations "in parallel."

Truth-table reductions are defined in [Rog67, HU79, Soa87], and weak truth-table reductions are defined in [Rog67, Soa87]. The following notation is standard [Soa87]:

Notation 2.1.1

- $A \leq_{\mathrm{m}} B$ if A is many-one reducible to B.
- $A \leq_{\mathrm{tt}} B$ if A is truth-table reducible to B.
- $A \leq_{\mathrm{wtt}} B$ if A is weak truth-table reducible to B.
- $A \leq_{\mathrm{T}} B$ if A is Turing reducible to B.

Informally, a truth-table (tt-) reduction from A to B works as follows: On input x, our machine M prepares a finite list of queries, makes the queries to B, plugs the oracle answers into a total recursive function, and outputs the result of the total recursive function, which must be equal to $\chi_A(x)$. A weak truth-table (wtt-) reduction from A to B works as follows: On input x, our machine M prepares a list of queries, makes the queries to B, performs an arbitrary computation using those oracle answers, and outputs the result of its computation, which must be equal to $\chi_A(x)$. The difference between a tt-reduction and a wtt-reduction is the following: If we are computing A via a tt-reduction to B then the computation must converge, *even if the computation receives incorrect oracle answers*, although the result of the computation is allowed to be incorrect; however, if we are computing A via a wtt-reduction to B then the computation is allowed to diverge if it receives incorrect oracle answers.

Lachlan [Lac65] has constructed an example that illustrates the difference between tt- and wtt-reductions: Let B be the union of two disjoint r.e. sets A and E. Then $A \leq_{\mathrm{wtt}} B$ by the following reduction, which makes only 1 query to B: First ask the oracle if $x \in B$. If $x \notin B$ then x cannot be in A, so reject. Otherwise, run the enumerators for the r.e. sets A and E, using a standard timesharing technique. Either $x \in A$ or $x \in E$, so eventually x is enumerated in one of the two sets. If x is enumerated in E then x cannot belong to A, so reject. If x is enumerated in A, then accept. There is no obvious tt-reduction from A to B, and in fact Lachlan's paper produces the sets A, B, and E via a priority argument that defeats every tt-reduction.

In most cases, when we refer to an oracle, we mean an oracle for a set. Therefore, we use the terms "oracle" and "set" interchangeably. Instead of writing "an oracle for B" this convention allows us to write simply "B." In a few instances, we need to refer to oracles that compute functions; we call such oracles "function oracles" in order to avoid confusion.

When no confusion can arise, we do not distinguish between sets and 0,1-valued total functions. Thus we identify the set A with its characteristic function χ_A.

On the other hand, we must distinguish between solving decision problems (*i.e.*, computing 0,1-valued functions, determining membership in a language) and computing functions. Many of the fundamental questions in this paper are more easily answered when they are asked about functions than when they are asked about decision problems.

We say that n queries to an oracle are made *in parallel*, or that n parallel queries are made, if a list of all n queries is formed before any of them is made.[2] Otherwise we say that n queries are made *in series*, or that n serial queries are made, or simply that n queries are made. The difference is that computation is allowed between serial queries to an oracle; the answer to an earlier query may determine what query is to be made next.

We define the *bounded query classes* relative to the oracle A:

[2] In [BK88], Book and Ko call parallel queries *nonadaptive queries*.

Definition 2.1.2

- $\mathrm{MQ}(n, A)$ is the set of machines with oracle A that make at most n queries to A.
- $\mathrm{FQ}(n, A)$ is the set of partial functions that are computable by a machine in $\mathrm{MQ}(n, A)$.
- $\mathrm{Q}(n, A)$ is the set of 0,1-valued total functions that are in $\mathrm{FQ}(n, A)$.
- $\mathrm{MQ}_{||}(n, A)$ is the set of machines with oracle A that make at most n queries to A, all queries being made in parallel.
- $\mathrm{FQ}_{||}(n, A)$ is the set of partial functions that are computable by a machine in $\mathrm{MQ}_{||}(n, A)$.
- $\mathrm{Q}_{||}(n, A)$ is the set of 0,1-valued total functions that are in $\mathrm{FQ}_{||}(n, A)$.

The preceding definitions make sense if the oracle A is replaced with a function oracle f. In subsequent sections, we assume that the bounded query classes, MQ, FQ, Q, $\mathrm{MQ}_{||}$, $\mathrm{FQ}_{||}$, and $\mathrm{Q}_{||}$, have been defined relative to function oracles as well as ordinary oracles.

The members of $\mathrm{MQ}(n, A)$ are called n-query A-machines. The members of $\mathrm{MQ}_{||}(n, A)$ are called n-parallel-query A-machines. Often we think of the oracle A as being extrinsic from the machine M that computes with it. If M computes with an unspecified oracle, we call M an oracle machine. When necessary to prevent confusion, we write $M^{()}$ to denote machine M with an unspecified oracle. We write M^A to denote the A-machine produced by equipping the oracle machine $M^{()}$ with an oracle for A.

If M^A is an n-query A-machine, it is not necessary that M^B be an n-query B-machine for all B, because the behavior of M is allowed to depend on the answers from the oracle. Suppose, for example, that, on input x, M computes the least $y > x$ such that y belongs to the oracle, using the obvious algorithm. If the oracle is equal to the set of natural numbers, then M only makes one query. However, if the oracle is equal to the empty set then M always makes infinitely many queries (and M does not even halt).

We can however, normalize, an n-query A-machine M^A, so that $M^{()}$ does not make more then n queries even when computing with an oracle other than A. To perform this normalization, we modify $M^{()}$ so that it uses a counter in order to keep track of the number of queries that it makes. If $M^{()}$ is about to make its $(n+1)$st query then we can have $M^{()}$ halt and reject (or print 0); alternatively we can have $M^{()}$ go into an infinite loop. Our choice of a particular normal behavior for $M^{()}$ will depend on our particular needs. The modifications to $M^{()}$ above do not effect the output of M^A. If $M^{()}$ has been normalized in one of these ways, then we call $M^{()}$ an n-query oracle machine. We can also normalize M so that M makes exactly n queries whenever M halts, by having M examine its counter before halting and make the necessary number of superfluous queries. This modification does not effect the output of M^B for any B.

Similarly, we can normalize an n-parallel-query A-machine M^A so that $M^{()}$ does not make more than one round of queries, even when computing with an oracle other than A. We can also guarantee that M makes exactly n parallel queries whenever M halts.

In contrast with A-machines, ordinary Turing machines (without oracle) will simply be called machines; however, when there is a possibility of confusion, ordinary Turing machines will be called $\emptyset$-machines.

Because we do not distinguish between sets and 0,1-valued total functions, we think of the elements of $\mathrm{Q}(n, A)$ as being sets, languages, decision problems, or 0,1-valued total functions according to our convenience.

We define reducibilities that use a bounded number of queries. When the reducibility requires only one query, we obtain an equivalence relation.

Definition 2.1.3

- A is *n-query reducible* to B (denoted $A \le_{n\text{-T}} B$) if $A \in \mathrm{Q}(n, B)$.

- A is *n-parallel-query reducible* to B (denoted $A \le_{n\text{-wtt}} B$) if $A \in \mathrm{Q}_{||}(n, B)$.

- A is 1-*query equivalent* to B (denoted $A \equiv_{\text{1-T}} B$) if $A \leq_{\text{1-T}} B$ and $B \leq_{\text{1-T}} A$.

Thus, n-query reducibility is a variant of Turing reducibility in which only n queries are allowed, and n-parallel-query reducibility is a variant of weak truth-table reducibility in which only n queries are allowed.

Definition 2.1.4 A is n-query complete for $\mathcal{C}$ if $A \in \mathcal{C}$ and $\mathcal{C} \subseteq \mathrm{Q}(n, A)$.

In other words A belongs to $\mathcal{C}$, and every set B belonging to $\mathcal{C}$ is n-query reducible to A. For example, the halting problem is 1-query complete for *r.e.* $\cup$ *co-r.e.*

The class of recursive sets (or total recursive 0,1-valued functions) is denoted by SREC, and the class of partial recursive functions is denoted by FREC.

We write $f \circ g$ to denote the composition of the functions f and g, so that $(f \circ g)(x) = f(g(x))$. We extend the definition of composition to apply to sets of functions:

Definition 2.1.5 If S_1 and S_2 are two sets of functions then

$$S_1 \circ S_2 = \{f_1 \circ f_2 \mid f_1 \in S_1 \text{ and } f_2 \in S_2\}.$$

In the next section we show that composition of two bounded serial query classes corresponds to allowing a number of queries to one oracle followed by a number of queries to a second oracle.

Definition 2.1.6 $f \| g$ denotes the concatenation of the functions f and g, as defined below:

i. If f and g are functions then

$$(f \| g)(x) = f(x), g(x).$$

(The comma (,) operator treats its two operands as lists and concatenates them. Scalar operands are treated as singleton lists. Thus the comma operator is associative.)

ii. If S_1 and S_2 are sets of functions then

$$S_1 \| S_2 = \{f_1 \| f_2 \mid (f_1 \in S_1) \text{ and } (f_2 \in S_2)\}.$$

We note that $\|$ is associative because the comma operator is associative.

In the next section we show how concatenation of two bounded parallel query classes is related to allowing a number of parallel queries to one oracle simultaneous with a number of parallel queries to a second oracle.

Relative to an oracle A we define two functions, F_n^A and $\#_n^A$, and two oracles, PARITY_n^A and GEQ^A. F_n^A, defined below, is a convenient notation for the results of n parallel queries to the oracle A.

Definition 2.1.7

$$\mathrm{F}_n^A(x_1, \ldots, x_n) = (\chi_A(x_1), \ldots, \chi_A(x_n)).$$

The function $\#_n^A$ determines how many of n strings are elements of A.

Definition 2.1.8

$$\#_n^A(x_1, \ldots, x_n) = \sum_{1 \le i \le n} \chi_A(x_i).$$

PARITY_n^A determines whether an odd number of n strings are elements of A.

Definition 2.1.9

$$\mathrm{PARITY}_n^A(x_1, \ldots, x_n) = \#_n^A(x_1, \ldots, x_n) \bmod 2.$$

GEQ^A determines whether at least t out of n strings are elements of A.

Definition 2.1.10

$$\mathrm{GEQ}^A(t; x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \#_n^A(x_1, \ldots, x_n) \ge t, \\ 0 & \text{otherwise.} \end{cases}$$

Following our general convention, we will frequently treat PARITY_n^A and GEQ^A as sets, rather than as 0,1-valued functions.

2.2 Observations about Q and FQ

Observation 2.2.1

i. $\mathrm{FQ}(1, A) = \mathrm{FQ}_{||}(1, A)$.

ii. $\mathrm{Q}(1, A) = \mathrm{Q}_{||}(1, A)$.

Proof: The definitions of n-serial- and n-parallel-query computation coincide when $n \le 1$. ∎

Observation 2.2.2 *If* $s < t$ *then*

i. $\mathrm{FQ}(s, A) \subseteq \mathrm{FQ}(t, A)$.

ii. $\mathrm{Q}(s, A) \subseteq \mathrm{Q}(t, A)$.

iii. $\mathrm{FQ}_{||}(s, A) \subseteq \mathrm{FQ}_{||}(t, A)$.

iv. $\mathrm{Q}_{||}(s, A) \subseteq \mathrm{Q}_{||}(t, A)$.

Proof: A computation that makes no more than s queries makes no more than t queries. ∎

Observation 2.2.3

i. If $\mathrm{FQ}(s, f) \subseteq \mathrm{FQ}(t, g)$ *then* $\mathrm{Q}(s, f) \subseteq \mathrm{Q}(t, g)$.

ii. If $\mathrm{FQ}_{||}(s, f) \subseteq \mathrm{FQ}_{||}(t, g)$ *then* $\mathrm{Q}_{||}(s, f) \subseteq \mathrm{Q}_{||}(t, g)$.

Proof: Let S be the set of 0,1-valued total functions (from the set of all strings to the set of all strings).

i. $\mathrm{Q}(s, f) = S \cap \mathrm{FQ}(s, f) \subseteq S \cap \mathrm{FQ}(t, g) = \mathrm{Q}(t, g)$.

ii. $\mathrm{Q}_{||}(s, f) = S \cap \mathrm{FQ}_{||}(s, f) \subseteq S \cap \mathrm{FQ}_{||}(t, g) = \mathrm{Q}_{||}(t, g)$.

∎

Observation 2.2.4 $\mathrm{F}_n^A \in \mathrm{FQ}_{||}(n, A)$.

Proof: A program to exhibit the answers to n given queries to A can make those n queries in parallel. ∎

Observation 2.2.5

i. If $f \in \mathrm{FQ}(s,g)$ *and* $g \in \mathrm{FQ}(t,h)$*, then* $f \in \mathrm{FQ}(st,h)$*.*

ii. If $f \in \mathrm{FQ}_{||}(s,g)$ *and* $g \in \mathrm{FQ}_{||}(t,h)$*, then* $f \in \mathrm{FQ}_{||}(st,h)$*.*

Proof:

i. We can compute f via an algorithm that makes s queries to g. We can answer each call to g by making t queries to h. Thus, we can compute f by making st queries to h.

ii. Similar to (i).

▮

Observation 2.2.6

i. $f \in \mathrm{FQ}(s,g)$ *if and only if* $\mathrm{FQ}(1,f) \subseteq \mathrm{FQ}(s,g)$*.*

ii. $f \in \mathrm{FQ}_{||}(s,g)$ *if and only if* $\mathrm{FQ}(1,f) \subseteq \mathrm{FQ}_{||}(s,g)$*.*

iii. $A \in \mathrm{Q}(s,g)$ *if and only if* $\mathrm{Q}(1,A) \subseteq \mathrm{Q}(s,g)$*.*

iv. $A \in \mathrm{Q}_{||}(s,g)$ *if and only if* $\mathrm{Q}(1,A) \subseteq \mathrm{Q}_{||}(s,g)$*.*

Proof:

i. Suppose that $h \in \mathrm{FQ}(1,f)$. By Observation 2.2.5, $h \in \mathrm{FQ}(s,g)$. Therefore $\mathrm{FQ}(1,f) \subseteq \mathrm{FQ}(s,g)$. Conversely, if $\mathrm{FQ}(1,f) \subseteq \mathrm{FQ}(s,g)$ then $f \in \mathrm{FQ}(1,f) \subseteq \mathrm{FQ}(s,g)$.

ii. Similar to (i).

iii. If $A \in \mathrm{FQ}(s,g)$ then $\mathrm{FQ}(1,A) \subseteq \mathrm{FQ}(s,g)$ by (i). By Observation 2.2.3, $\mathrm{Q}(1,A) \subseteq \mathrm{Q}(s,g)$. Conversely, if $\mathrm{Q}(1,A) \subseteq \mathrm{Q}(s,g)$ then $A \in \mathrm{Q}(1,A) \subseteq \mathrm{Q}(s,g)$.

iv. Similar to (iii).

▮

Observation 2.2.7 $\mathrm{FQ}(1, \mathrm{F}_n^A) = \mathrm{FQ}_{||}(n, A)$.

Proof: By Observation 2.2.4, $\mathrm{F}_n^A \in \mathrm{FQ}_{||}(n, A)$. Therefore, by Observation 2.2.6, $\mathrm{FQ}(1, \mathrm{F}_n^A) \subseteq \mathrm{FQ}_{||}(n, A)$. Conversely, let $f \in \mathrm{FQ}_{||}(n, A)$. Then f can be computed by an algorithm that makes only n parallel queries to A. Therefore f can be computed by an algorithm that makes one call to a function that answers n parallel queries to A. ∎

Observation 2.2.8

i. $\#_n^A \in \mathrm{FQ}_{||}(n, A)$.

ii. $\mathrm{PARITY}_n^A \in \mathrm{Q}_{||}(n, A)$.

Proof: Both can be computed by making n parallel queries to A. ∎

Observation 2.2.9 $\#_n^A \in \mathrm{FQ}(1, \#_n^{\bar{A}})$.

Proof: $\#_n^A(x_1, \ldots, x_n) = n - \#_n^{\bar{A}}(x_1, \ldots, x_n)$. ∎

Observation 2.2.10 *If* $A \leq_{\mathrm{m}} B$ *then*

i. $A \in \mathrm{Q}(1, B)$.

ii. $(\forall n)[\mathrm{FQ}(n, A) \subseteq \mathrm{FQ}(n, B)]$.

iii. $(\forall n)[\mathrm{FQ}_{||}(n, A) \subseteq \mathrm{FQ}_{||}(n, B)]$.

iv. $\mathrm{PARITY}_n^A \leq_{\mathrm{m}} \mathrm{PARITY}_n^B$.

v. $\#_n^A \in \mathrm{FQ}(1, \#_n^B)$.

Proof: Since $A \leq_{\mathrm{m}} B$, let f be a total recursive function such that $x \in A$ if and only if $f(x) \in B$.

i. An m-reduction requires only one query.

ii. We can modify any n-query A-machine so that instead of querying whether $x \in A$, it queries whether $f(x) \in B$. The new machine is an n-query B-machine that computes the same function.

iii. Similar to (ii).

iv. $(x_1, \ldots, x_n) \in \text{PARITY}_n^A$ if and only if $(f(x_1), \ldots, f(x_n)) \in \text{PARITY}_n^B$.

v. $\#_n^A(x_1, \ldots, x_n) = \#_n^B(f(x_1), \ldots, f(x_n))$.

▮

Observation 2.2.11 *A function g can be computed by making at most n_1 parallel queries to f_1, followed by at most n_2 parallel queries to f_2, … followed by at most n_r parallel queries to f_r if and only if*

$$g \in \text{FQ}_{||}(n_r, f_r) \circ \text{FQ}_{||}(n_{r-1}, f_{r-1}) \circ \cdots \circ \text{FQ}_{||}(n_1, f_1).$$

Proof: First, assume that g can be computed by an oracle Turing machine that makes at most n_1 parallel queries to f_1, followed by at most n_2 parallel queries to f_2, … followed by at most n_r parallel queries to f_r. Without loss of generality, we assume that g's output is stored on a special buffer tape that is printed as part of the halt instruction (this prevents the output of g from interfering with the input/output relations of the r functions that we are composing).

We compute a function g_0 as follows: output the initial instantaneous description of g (*i.e.*, the starting tape configuration and the starting state).

For $1 \leq i \leq r$, we compute a function g_i as follows: The input to g_i is an instantaneous description of a computation. Simulate g, starting from the given instantaneous description, until g is about to halt or make some parallel queries. If g is about to make no more than n_i parallel queries to f_i, continue the simulation until after the queries are made, and then output the instantaneous description of g. Otherwise, stop simulating, and then output the instantaneous description.

We compute a function g_{r+1} as follows: The input to g_{r+1} is an instantaneous description of a computation. Simulate g, starting from the given instantaneous description, until g is about to halt or make some queries. In either case, halt.

For $1 \leq i \leq r$, $g_i \in \mathrm{FQ}_{||}(n_i, r_i)$. Because g_0 and g_{r+1} are recursive, $g_1 \circ g_0 \in \mathrm{FQ}_{||}(n_1, f_r)$ and $g_{r+1} \circ g_r \in \mathrm{FQ}_{||}(n_r, f_r)$. Therefore

$$\begin{aligned} g &= (g_{r+1} \circ g_r) \circ g_{r-1} \circ \cdots \circ (g_1 \circ g_0) \\ &\in \mathrm{FQ}_{||}(n_r, f_r) \circ \mathrm{FQ}_{||}(n_{r-1}, f_{r-1}) \circ \cdots \circ \mathrm{FQ}_{||}(n_1, f_1). \end{aligned}$$

Conversely, assume that $g = g_r \circ g_{r-1} \circ \cdots \circ g_1$, where each g_i is a function in $\mathrm{FQ}_{||}(n_i, f_i)$. We evaluate g_1 and use its output as the input to g_2, then we evaluate g_2 and use its output as the input to g_3, ... and then finally we evaluate g_r. That algorithm computes $g(x)$ by making n_1 parallel queries to f_1, followed by n_2 parallel queries to f_2, ... followed by n_r parallel queries to f_r. ▮

Corollary 2.2.12

$$\mathrm{FQ}(a+b, f) = \mathrm{FQ}(a, f) \circ \mathrm{FQ}(b, f).$$

Proof: By Observation 2.2.11, for every n

$$\mathrm{FQ}(n, f) = \underbrace{\mathrm{FQ}_{||}(1, f) \circ \cdots \circ \mathrm{FQ}_{||}(1, f)}_{n}.$$

Therefore

$$\begin{aligned} \mathrm{FQ}(a+b, f) &= \underbrace{\mathrm{FQ}_{||}(1, f) \circ \cdots \circ \mathrm{FQ}_{||}(1, f)}_{a+b} \\ &= \underbrace{\mathrm{FQ}_{||}(1, f) \circ \cdots \circ \mathrm{FQ}_{||}(1, f)}_{a} \circ \underbrace{\mathrm{FQ}_{||}(1, f) \circ \cdots \circ \mathrm{FQ}_{||}(1, f)}_{b} \\ &= \mathrm{FQ}(a, f) \circ \mathrm{FQ}(b, f) \qquad \text{by Observation 2.2.11} \end{aligned}$$

▮

Observation 2.2.13 *The function g can be computed by making at most n_1 parallel queries to f_1, simultaneous with at most n_2 parallel queries to $f_2, \ldots$ simultaneous with at most n_r simultaneous queries to f_r if and only if*

$$g \in \mathrm{FREC} \circ (\mathrm{FQ}_{||}(n_1, f_1) \parallel \mathrm{FQ}_{||}(n_2, f_2) \parallel \cdots \parallel \mathrm{FQ}_{||}(n_r, f_r)).$$

Proof: First, assume that g can be computed by an oracle Turing machine that makes at most n_1 parallel queries to f_1, simultaneous with at most n_2 parallel queries to f_2, … simultaneous with at most n_r simultaneous queries to f_r. We assume that g is normalized so that g makes exactly n_1 parallel queries to f_1, simultaneous with exactly n_2 parallel queries to f_2, … simultaneous with exactly n_r simultaneous queries to f_r, whenever g converges.

We compute a function g_i as follows: Simulate g until g is about to make its parallel queries, make the n_i parallel queries to f_i, and output the results of those n_i queries.

We compute a function h as follows: The input to h consists of the input to g followed by a sequence of $\sum_{1\leq i\leq r} n_i$ oracle answers. Simulate g using the oracle answers given by the input sequence, rather than making any queries. Then

$$\begin{aligned} g &= h \circ (g_1 \,\|\, g_2 \,\|\, \cdots \,\|\, g_r) \\ &\in \mathrm{FREC} \circ (\mathrm{FQ}_{||}(n_1, f_1) \,\|\, \mathrm{FQ}_{||}(n_2, f_2) \,\|\, \cdots \,\|\, \mathrm{FQ}_{||}(n_r, f_r)). \end{aligned}$$

Conversely, assume that

$$g = h \circ (g_1 \,\|\, \cdots \,\|\, g_r),$$

where h is partial recursive, and each g_i is a function in $\mathrm{FQ}_{||}(n_i, f_i)$. We assume that g_i is normalized so that it makes exactly n_i parallel queries whenever it halts. Then the following algorithm computes $g(x)$ by making n_1 parallel queries to f_1, simultaneous with n_2 parallel queries to f_2, … simultaneous with n_r parallel queries to f_r: Simulate g_1 through g_r, and suspend each of them right before it is about to make its oracle queries. When each of g_1 through g_r is ready to make its oracle queries, continue the simulation, making all queries simultaneously. When g_1 through g_r have terminated, simulate h, and print h's answer. ∎

Corollary 2.2.14

$$\mathrm{FQ}_{||}(a+b, f) = \mathrm{FREC} \circ (\mathrm{FQ}_{||}(a, f) \,\|\, \mathrm{FQ}_{||}(b, f)).$$

Proof: Let $n_1 = a$, $n_2 = b$, and $f_1 = f_2 = f$ in Observation 2.2.13. ∎

Observation 2.2.15

i. If $\mathrm{FQ}(n+1,B)=\mathrm{FQ}(n,B)$ *then*

$$(\forall m\geq n)[\mathrm{FQ}(m,B)=\mathrm{FQ}(n,B)].$$

ii. If $\mathrm{FQ}_{||}(n+1,B)=\mathrm{FQ}_{||}(n,B)$ *then*

$$(\forall m\geq n)[\mathrm{FQ}_{||}(m,B)=\mathrm{FQ}_{||}(n,B)].$$

Proof:

i. Assume that $\mathrm{FQ}(n,B)=\mathrm{FQ}(n+1,B)$. For all $t\geq 0$,

$$\begin{aligned}\mathrm{FQ}(n+t+1,B) &= \mathrm{FQ}(n+1+t,B)\\ &= \mathrm{FQ}(n+1,B)\circ\mathrm{FQ}(t,B) \qquad \text{by Corollary 2.2.12}\\ &= \mathrm{FQ}(n,B)\circ\mathrm{FQ}(t,B) \qquad \text{by assumption}\\ &= \mathrm{FQ}(n+t,B) \qquad \text{by Corollary 2.2.12.}\end{aligned}$$

Thus $\mathrm{FQ}(n+t+1,B)=\mathrm{FQ}(n+t,B)$ for all $t\geq 0$. By transitivity, $\mathrm{FQ}(m,B)=\mathrm{FQ}(n,B)$ for all $m\geq n$.

ii. Assume that $\mathrm{FQ}_{||}(n,B)=\mathrm{FQ}_{||}(n+1,B)$. For all $t\geq 0$,

$$\begin{aligned}\mathrm{FQ}_{||}(n+t+1,B) &= \mathrm{FQ}_{||}(n+1+t,B)\\ &= \mathrm{FREC}\circ(\mathrm{FQ}_{||}(n+1,B)\,\|\,\mathrm{FQ}_{||}(t,B)) \qquad \text{by Corollary 2.2.14}\\ &= \mathrm{FREC}\circ(\mathrm{FQ}_{||}(n,B)\,\|\,\mathrm{FQ}_{||}(t,B)) \qquad \text{by assumption}\\ &= \mathrm{FQ}_{||}(n+t,B) \qquad \text{by Corollary 2.2.14.}\end{aligned}$$

Thus $\mathrm{FQ}_{||}(n+t+1,B)=\mathrm{FQ}_{||}(n+t,B)$ for all $t\geq 0$. By transitivity, $\mathrm{FQ}_{||}(m,B)=\mathrm{FQ}_{||}(n,B)$ for all $m\geq n$.

▮

Observation 2.2.16 *Let $n_1, \ldots, n_r$ be nonnegative integers, let $s = \max_{1 \le i \le r} n_i$, and let*

$$m_j = |\{n_i \mid n_i \ge j\}|.$$

Then

$$\mathrm{FQ}(n_1, A) \parallel \cdots \parallel \mathrm{FQ}(n_r, A) \subseteq \mathrm{FQ}_{||}(m_s, A) \circ \mathrm{FQ}_{||}(m_{s-1}, A) \circ \cdots \circ \mathrm{FQ}_{||}(m_1, A).$$

Proof: If $f \in \mathrm{FQ}(n_1, A) \parallel \cdots \parallel \mathrm{FQ}(n_r, A)$ then we can compute f by timesharing an n_1-query A-machine, an n_2-query A-machine, ... and an n_r-query A-machine. Without loss of generality, we assume that each n_i-query A-machine makes exactly n_i queries whenever it halts. We force those machines to synchronize their queries; thus f is computed by a machine that makes m_1 parallel queries to A, followed by m_2 parallel queries to A, ... followed by m_s parallel queries to A. ∎

Observation 2.2.17

$$\underbrace{\mathrm{FQ}(n, A) \parallel \cdots \parallel \mathrm{FQ}(n, A)}_{m} \subseteq \underbrace{\mathrm{FQ}_{||}(m, A) \circ \cdots \circ \mathrm{FQ}_{||}(m, A)}_{n}$$

Proof: Let $r = m$ and let $n_1 = n_2 = \cdots = n_r = n$ in Observation 2.2.16. Then $s = n$ and $m_1 = m_2 = \cdots = m_s = m$. ∎

Observation 2.2.18 $\#_n^A \in \mathrm{FQ}(\lceil \log (n+1) \rceil, \mathrm{GEQ}^A)$.

Proof: $\#_n^A(x_1, \ldots, x_n)$ is an integer k such that $0 \le k \le n$. For any t, a single query to GEQ^A will tell us whether $k \ge t$. Thus, a binary search determines k by making $\lceil \log (n+1) \rceil$ queries to GEQ^A. ∎

The following generalization of Observation 2.2.18 is key to the classification in Chapter 3 of functions computable by machines that make several rounds of parallel queries to an oracle for the halting problem.

Observation 2.2.19

$$\#^A_{(n_1+1)\cdots(n_r+1)-1} \in \mathrm{FQ}_{||}(n_r, \mathrm{GEQ}^A) \circ \mathrm{FQ}_{||}(n_{r-1}, \mathrm{GEQ}^A) \circ \cdots \circ \mathrm{FQ}_{||}(n_1, \mathrm{GEQ}^A).$$

Proof:

Let $N = (n_1 + 1) \cdots (n_r + 1)$. The value taken on by $\#^A_{N-1}$ is an integer k such that $0 \le k \le N - 1$; thus k has one of N possible values. For any t, a single query to GEQ^A will tell us whether $k \ge t$. With n_1 parallel queries, we ask whether $k \ge N/(n_1 + 1)$, $k \ge 2N/(n_1 + 1)$, $\ldots$, $k \ge n_1 N/(n_1 + 1)$. These queries restrict k to a range of $N/(n_1 + 1)$ possible values. Similarly, the next n_2 parallel queries can restrict k to a range of $N/((n_1+1)(n_2+1))$ possible values. We continue in this way, until the final n_r parallel queries restrict k to a range of $N/((n_1+1)\cdots(n_r+1)) = 1$ possible value. Thus $\#^A_{N-1}$ can be computed by making n_1 parallel queries to GEQ^A, followed by n_2 parallel queries to GEQ^A, ... followed by n_r parallel queries to GEQ^A. By Observation 2.2.11,

$$\#^A_{N-1} \in \mathrm{FQ}_{||}(n_r, \mathrm{GEQ}^A) \circ \mathrm{FQ}_{||}(n_{r-1}, \mathrm{GEQ}^A) \circ \cdots \circ \mathrm{FQ}_{||}(n_1, \mathrm{GEQ}^A).$$

∎

In Chapter 3, we will use the following observation to show that

$$\mathrm{Q}_{||}(n, K) \subset \mathrm{Q}_{||}(n+1, K),$$

where K is an oracle for the halting problem.

Observation 2.2.20

$$\#^A_{2n+1} \in \mathrm{FREC} \circ (\mathrm{FQ}(1, \#^{\mathrm{GEQ}^A}_n) \parallel \mathrm{FQ}(1, \mathrm{PARITY}^{\mathrm{GEQ}^A}_{n+1})).$$

Proof: Suppose that we are to compute $\#^A_{2n+1}(\vec{x})$, where $\vec{x} = (x_1, \ldots, x_{2n+1})$. Let $\vec{y} = (y_1, \ldots, y_{2n+1})$, where

$$y_i = (i; \vec{x}).$$

Then

$$\#^A_{2n+1}(\vec{x}) = \#^{\mathrm{GEQ}^A}_{2n+1}(\vec{y}).$$

Let

$$\begin{aligned} z &= \#^{\mathrm{GEQ}^A}_{2n+1}(\vec{y}), \\ t &= \#^{\mathrm{GEQ}^A}_{n}(y_2, y_4, \ldots, y_{2n}), \\ p &= \mathrm{PARITY}^{\mathrm{GEQ}^A}_{n+1}(y_1, y_3, \ldots, y_{2n+1}). \end{aligned}$$

Then $z = 2t$ or $z = 2t + 1$, depending on whether $y_{2t+1} \in \mathrm{GEQ}^A$. Since the parity function computed above changes value if $y_{2t+1} \in \mathrm{GEQ}^A$, the value of p determines whether $z = 2t$ or $z = 2t + 1$; thus z is determined by the values of t and p. In fact

$$z = \begin{cases} 2t + p & \text{if } t \text{ is even} \\ 2t + 1 - p & \text{otherwise.} \end{cases}$$

Thus, we can compute $\#^A_{2n+1}(\vec{x}\,) = \#^{\mathrm{GEQ}^A}_{2n+1}(\vec{y}\,)$ by making one query to $\#^{\mathrm{GEQ}^A}_n$ simultaneous with one query to $\mathrm{PARITY}^{\mathrm{GEQ}^A}_{n+1}$. ∎

A version of Kleene's recursion theorem is true for k-query A-machines. We will use the following notation exclusively in connection with the recursion theorem.

Notation 2.2.21

i. φ^A_e is the function computed by machine e relative to oracle A.

ii.

$$\varphi^{A\leq k}_e(x) = \begin{cases} \varphi^A_e(x) & \text{if } \varphi^A_e(x) \text{ converges after making at most } k \text{ queries} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Note that $\varphi^{A\leq k}_e \in \mathrm{FQ}(k, A)$.

Observation 2.2.22

i. If f is a total recursive mapping from $\mathrm{MQ}(k, A)$ to $\mathrm{MQ}(k, A)$, then there exists a machine $n \in \mathrm{MQ}(k, A)$ such that $\varphi^A_n = \varphi^A_{f(n)}$.

ii. If f is a total recursive mapping from $\mathrm{MQ}_{||}(k, A)$ to $\mathrm{MQ}_{||}(k, A)$, then there exists a machine $n \in \mathrm{MQ}_{||}(k, A)$ such that $\varphi^A_n = \varphi^A_{f(n)}$.

Proof:

i. We prove this by making minor changes to the proof in [Soa87] of the ordinary recursion theorem. By the s-m-n theorem, there is a total recursive function d such that

$$(\forall v, z)[\varphi^A_{d(v)}(z) = \varphi^{A\leq k}_{\varphi_v(v)}(z)].$$

Choose v such that $\varphi_v = f \circ d$; then

$$(\forall z)[\varphi^A_{d(v)}(z) = \varphi^{A\leq k}_{f\circ d(v)}(z)].$$

Let $n = d(v)$. Then

$$(\forall z)[\varphi^A_n(z) = \varphi^{A\leq k}_{f(n)}(z)].$$

By the definition of d, machine $n = d(v)$ is a k-query A-machine. Therefore, by the definition of f, machine $f(n)$ must also be a k-query A-machine. Therefore, $\varphi^{A\leq k}_{f(n)} = \varphi^A_{f(n)}$, so

$$(\forall z)[\varphi^A_n(z) = \varphi^A_{f(n)}(z)].$$

Thus, n is a fixed point of f.

ii. Similar to (i).

∎

Chapter 3

Bounded Queries to the Halting Problem

In this chapter, we study the classes of sets and functions computable by machines that make a bounded number of queries to an oracle for the halting problem. In Chapter 4, we will generalize some of these results to arbitrary nonrecursive oracles.

3.1 Lemmas About K

We use K to denote the halting problem, *i.e.*, the set of machines that halt on empty input. Since the usual variants of the halting problem are recursively isomorphic (see, for example, [Soa87]), we lose no generality by considering only this version of the halting problem. We exhibit some straightforward properties of K.

Lemma 3.1.1 *If A is r.e. then* GEQ^A *is r.e.*

Proof: Given a nondeterministic acceptor for A, we accept GEQ^A as follows: On input $(t; x_1, \ldots, x_n)$, run the nondeterministic acceptor for A on each of x_1 through x_n, keeping track of how many are accepted. If at least t of them are accepted, then accept; otherwise reject. This nondeterministic algorithm accepts GEQ^A. ∎

Lemma 3.1.2 $\mathrm{GEQ}^K \equiv_{\mathrm{m}} K$.

Proof: Because K is r.e., GEQ^K is r.e. by Lemma 3.1.1. Therefore $\mathrm{GEQ}^K \leq_{\mathrm{m}} K$. Conversely, $x \in K$ if and only $(1; x) \in \mathrm{GEQ}^K$, so $K \leq_{\mathrm{m}} \mathrm{GEQ}^K$. ∎

The next lemma shows that we can determine how many of n programs halt by asking only $\lceil \log(n+1) \rceil$ queries to K.

Lemma 3.1.3 $\#_n^K \in \mathrm{FQ}(\lceil \log(n+1) \rceil, K)$.

Proof: By Observation 2.2.18,

$$\#_n^K \in \mathrm{FQ}(\lceil \log(n+1) \rceil, \mathrm{GEQ}^K).$$

By Lemma 3.1.2, $\mathrm{GEQ}^K \equiv_{\mathrm{m}} K$. Therefore,

$$\mathrm{FQ}(\lceil \log(n+1) \rceil, \mathrm{GEQ}^K) = \mathrm{FQ}(\lceil \log(n+1) \rceil, K),$$

by Observation 2.2.10(ii). Therefore,

$$\#_n^K \in \mathrm{FQ}(\lceil \log(n+1) \rceil, K).$$

∎

In order to determine which of n numbers belong to an r.e. set B, we only need to know how many of them belong to the set B, as shown by the next lemma.

Lemma 3.1.4 *If B is r.e. then* $\mathrm{F}_n^B \in \mathrm{FQ}(1, \#_n^B)$.

Proof: Here is an algorithm relative to $\#_n^B$ to determine which of $x_1, \ldots, x_n$ belong to B: Let $t = \#_n^K(x_1, \ldots, x_n)$. Simulate an enumerator for B until at least t of the numbers $x_1, \ldots, x_n$ have been enumerated. If x_i has been enumerated by that time, then $x_i \in B$; otherwise $x_i \notin B$. ∎

Lemma 3.1.5

i. $\mathrm{F}_n^K \in \mathrm{FQ}(1, \#_n^K)$.

ii. $\mathrm{FQ}_{||}(n, K) = \mathrm{FQ}(1, \#_n^K)$.

Proof:

i. This follows from Lemma 3.1.4 because K is r.e.

ii. By Observation 2.2.7, $\mathrm{FQ}_{||}(n, K) = \mathrm{FQ}(1, \mathrm{F}_n^K)$. By (i) and Observation 2.2.6(i), $\mathrm{FQ}(1, \mathrm{F}_n^K) \subseteq \mathrm{FQ}(1, \#_n^K)$. Therefore, $\mathrm{FQ}_{||}(n, K) \subseteq \mathrm{FQ}(1, \#_n^K)$. Conversely, $\#_n^K \in \mathrm{FQ}_{||}(n, K)$, so $\mathrm{FQ}(1, \#_n^K) \subseteq \mathrm{FQ}_{||}(n, K)$ by Observation 2.2.6(i).

∎

The next lemma shows that we can determine which of n programs halt by asking only $\lceil \log (n + 1) \rceil$ serial queries to K.

Lemma 3.1.6

i. $\mathrm{F}_n^K \in \mathrm{FQ}(\lceil \log (n + 1) \rceil, K)$.

ii. $\mathrm{FQ}_{||}(n, K) \subseteq \mathrm{FQ}(\lceil \log (n + 1) \rceil, K)$.

Proof:

i. By Lemma 3.1.5,
$$\mathrm{F}_n^K \in \mathrm{FQ}(1, \#_n^K).$$
By Lemma 3.1.3,
$$\#_n^K \in \mathrm{FQ}(\lceil \log (n + 1) \rceil, K).$$
Therefore, by Observation 2.2.5(i),
$$\mathrm{F}_n^K \in \mathrm{FQ}(\lceil \log (n + 1) \rceil, K).$$

ii. By Observation 2.2.7, $\mathrm{FQ}_{||}(n, K) = \mathrm{FQ}(1, \mathrm{F}_n^K)$. By (i) and Observation 2.2.6(i),
$$\mathrm{FQ}(1, \mathrm{F}_n^K) \subseteq \mathrm{FQ}(\lceil \log (n + 1) \rceil, K).$$

∎

Lemma 3.1.6(ii) states a relationship between serial queries and parallel queries to K. In the next section we will prove a converse to Lemma 3.1.6(ii).

Lemma 3.1.5 allows us to replace n parallel queries to K by a single query to $\#_n^K$. The next lemma shows how to transform a query to $\#_n^K$ into a special form.

Lemma 3.1.7 *There is a total recursive function $\vec{y}$ such that for every natural number n and every n-tuple $\vec{x} = (x_1, \ldots, x_n)$*

$$\#_n^K(\vec{y}(\vec{x})) = \max\{i \mid y_i \in K\} = \#_n^K(\vec{x}),$$

where $\vec{y} = (y_1, \ldots, y_n)$.

Proof: By Lemma 3.1.1, GEQ^K is r.e. Therefore, there exists a total recursive function f such that $z \in \mathrm{GEQ}^K$ if and only if $f(z) \in K$. Let $y_i = f(i; \vec{x})$. If $(i+1; \vec{x}) \in \mathrm{GEQ}^K$ then $(i; \vec{x}) \in \mathrm{GEQ}^K$, so if $y_{i+1} \in K$ then $y_i \in K$. Therefore, $\chi_K(y_i) \geq \chi_K(y_{i+1})$. Because of this monotonicity condition,

$$\begin{aligned}\#_n^K(\vec{y}) &= \max\{i \mid y_i \in K\}\\ &= \max\{i \mid (i; \vec{x}) \in \mathrm{GEQ}^K\}\\ &= \max\{i \mid \#_n^K(\vec{x}) \geq i\}\\ &= \#_n^K(\vec{x}).\end{aligned}$$

∎

Lemma 3.1.7 allows us to replace n parallel queries to K with n queries to K in such a way that the answers to the queries are monotone. We will use that transformation explicitly in the remainder of this chapter, instead of referring to the lemma.

The following lemma depends only on the fact that $\mathrm{GEQ}^K \leq_{\mathrm{m}} K$.

Lemma 3.1.8

$$\#_{2n+1}^K \in \mathrm{FREC} \circ (\mathrm{FQ}(1, \#_n^K) \parallel \mathrm{FQ}(1, \mathrm{PARITY}_{n+1}^K)).$$

Proof: By Observation 2.2.20,

$$\#^K_{2n+1} \in \mathrm{FREC} \circ (\mathrm{FQ}(1, \#^{\mathrm{GEQ}^K}_n) \parallel \mathrm{FQ}(1, \mathrm{PARITY}^{\mathrm{GEQ}^K}_{n+1})).$$

By Lemma 3.1.2, $\mathrm{GEQ}^K \equiv_{\mathrm{m}} K$. Therefore,

$$\mathrm{FQ}(1, \#^{\mathrm{GEQ}^K}_n) = \mathrm{FQ}(1, \#^K_n)$$

by Observation 2.2.10(v); and

$$\mathrm{FQ}(1, \mathrm{PARITY}^{\mathrm{GEQ}^K}_n) = \mathrm{FQ}(1, \mathrm{PARITY}^K_n)$$

by Observation 2.2.10(iv) and Observation 2.2.10(ii). Therefore,

$$\#^K_{2n+1} \in \mathrm{FREC} \circ (\mathrm{FQ}(1, \#^K_n) \parallel \mathrm{FQ}(1, \mathrm{PARITY}^K_{n+1})).$$

▮

We will use Lemma 3.1.8 in the next section to show that $\mathrm{Q}_{||}(n, K)$ is a proper subset of $\mathrm{Q}_{||}(n+1, K)$.

3.2 Separating the Bounded Query Classes

Lemma 3.2.1 $\mathrm{FQ}(n, K) \subseteq \mathrm{FQ}_{||}(2^n - 1, K)$.

Proof: We show how to simulate an n-query K-machine M by a (2^n-1)-parallel-query K-machine. Let $M \in \mathrm{MQ}(n, K)$. Regardless of the oracle that M uses, we know that there are at most 2^{i-1} possibilities for the ith query — one for each sequence of answers to previous $i-1$ queries. Thus we have an *a priori* bound of $2^n - 1$ different queries that could be made, regardless of the answers given by the oracle. It is not in general possible to pre-compute what all these queries might be, because some purported sequence of oracle answers might force M into a non-terminating computation.

However, we can construct a query that has the same answer as the ith query if the ith query is actually made (the answer is irrelevant if the ith query is not made). For each Boolean sequence s of $i-1$ potential oracle answers, we construct a $\emptyset$-machine M^s that computes as follows: Using the sequence s to answer the first $i-1$ queries, simulate M until M produces its ith query (go into an infinite loop if M halts before producing i queries); then simulate M's ith query q (until q halts) by using the universal Turing machine; and then halt. M^s makes no queries; furthermore, if the assumed sequence of $i-1$ oracle answers is correct, then M^s halts if and only if M makes at least i queries and M's ith query belongs to K.

In other words, for each sequence of potential answers to the first $i-1$ queries, we have shown how to produce a query (namely "does M^s halt?") that has the same answer as M's ith query if the first $i-1$ answers are correct and if M actually makes at least i queries. If the first $i-1$ answers are not all correct or if M makes fewer than i queries, we do not care about the answer to the query that we produce.

By determining whether each machine M^s halts, we determine the answers to all of M's possible queries. The following algorithm simulates M by making only $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ queries to K: For each sequence s of fewer than n bits, query K to determine whether M^s halts. Simulate M, by substituting known answers for all of M's queries to K. Thus $\mathrm{FQ}(n,K) \subseteq \mathrm{FQ}_{||}(2^n-1,K)$. ▮

Theorem 3.2.2 $\mathrm{FQ}(n,K) = \mathrm{FQ}_{||}(2^n-1,K)$.

Proof: By Lemma 3.1.6(ii),

$$(\forall n)[\mathrm{FQ}_{||}(n,K) \subseteq \mathrm{FQ}(\lceil \log{(n+1)} \rceil, K)].$$

By replacing n with 2^n-1 in the previous statement, we obtain

$$\mathrm{FQ}_{||}(2^n-1,K) \subseteq \mathrm{FQ}(n,K).$$

Conversely, by Lemma 3.2.1,

$$\mathrm{FQ}(n,K) \subseteq \mathrm{FQ}_{||}(2^n-1,K).$$

▮

Theorem 3.2.7 below states that $\mathrm{Q}(n, K) \subset \mathrm{Q}(n+1, K)$. We prove the theorem by showing that the halting problem for n-query K-machines is in $\mathrm{Q}(n+1, K)$ but not in $\mathrm{Q}(n, K)$.

Given a machine M^B in $\mathrm{MQ}(n, B)$ we can modify $M^{()}$ (in a fixed way) so that $M^{()}$ never makes more than n queries to its oracle (by having M keep count of how many queries it makes). Such a machine is said to be in standard form. It is important that we choose a fixed way to modify M so that an algorithm can check whether a machine is in standard form.

Notation 3.2.3

- $\mathrm{MQ}^*(n, A)$ is the set of machines in $\mathrm{MQ}(n, A)$ that are in standard form.
- $\mathrm{MQ}^*_{||}(n, A)$ is the set of machines in $\mathrm{MQ}_{||}(n, A)$ that are in standard form.

Definition 3.2.4 If $\mathcal{C}$ is a set of machines then $H_{\mathcal{C}}$ is the halting problem for $\mathcal{C}$. That is,

$$\mathrm{H}_{\mathcal{C}} = \{x \in \mathcal{C} \mid x \text{ halts on empty input}\}.$$

Informally, we call $\mathrm{H}_{\mathrm{MQ}^*(n,B)}$ the halting problem for n-query B-machines.

Lemma 3.2.5 *For every set B and natural number n*

i. $\mathrm{H}_{\mathrm{MQ}^*(n,B)} \notin \mathrm{Q}(n, B)$.

ii. $\mathrm{H}_{\mathrm{MQ}^*_{||}(n,B)} \notin \mathrm{Q}_{||}(n, B)$.

Proof:

i. (This is analogous to the standard proof that the halting problem for $\emptyset$-machines is unsolvable.) Given a machine x and an input y we can build another machine that ignores its input and simulates machine x on input y (by the s-m-n Theorem). Thus the set of all pairs (x, y) such that the n-query B-machine x halts on input y is m-reducible to $\mathrm{H}_{\mathrm{MQ}^*(n,B)}$. Therefore it suffices to show that this more general halting problem is not solvable by any n-query B-machine. Suppose that M were an n-query B-machine that could solve the halting problem for n-query B-machines on arbitrary input; that is, if x is an n-query B-machine in standard form then

$$M(x,y) = \begin{cases} \text{true} & \text{if program } x \text{ halts on input } y \\ \text{false} & \text{otherwise.} \end{cases}$$

We define a new machine u such that

$$u(x) \begin{cases} \text{goes into an infinite loop} & \text{if } M(x,x) = \text{true} \\ \text{halts} & \text{otherwise.} \end{cases}$$

Clearly u is an n-query B-machine in standard form. By construction, u fails to halt on input u if and only if u halts on input u. That is a contradiction.

ii. Simular to (i).

∎

A different proof is possible, via the Recursion Theorem (2.2.22) for n-query B-machines (note that Theorem 2.2.22 is easily extended to machines in standard form): Assume that M is an n-query B-machine that solves the halting problem for n-query B-machines. Define an n-query B-machine program u such that for all x

$$\varphi_u^B(x) = \begin{cases} \text{undefined} & \text{if } M(u) = \text{true} \\ 0 & \text{otherwise.} \end{cases}$$

(See [Soa87, pages 34–35] for a justification of this informal use of the Recursion Theorem.) Then φ_u^B converges on empty input if and only if φ_u^B diverges on empty input. The proof for n-parallel-query B-machines is similar.

A special case of the previous lemma is that n queries to K do not allow us to solve the halting problem for n-query K-machines. However, $n+1$ queries to K *do* allow us to solve the halting problem for n-query K-machines, as shown below.

Lemma 3.2.6 $\mathrm{H}_{\mathrm{MQ}^*(n,K)} \in \mathrm{Q}(n+1,K)$.

Proof: Suppose that we are to determine whether an n-query machine x in standard form halts on empty input. The problem would be trivial if x always made exactly n queries to K. If that were the case, we would simulate x until x had made its n queries, and then we would ask a final query to K in order to determine if the remainder of x's computation would terminate. However, if x diverges there is no guarantee that x uses its full allotment of n queries to K.

We avoid that pitfall as follows: For $1 \leq i \leq n+1$ we define a machine x_i that uses the answers to the first $i-1$ queries (if that many queries are actually made) in order to simulate x until x has halted or is about to make another query. If x has halted then x_i halts; otherwise x_i simulates x's ith query q_i (until q_i halts) by using the universal Turing machine, and then x_i halts. Thus x_i halts if and only if (1) x halts without making i queries or (2) x makes at least i queries and the ith query made by x is in K.

The following algorithm determines whether an n-query K-machine halts on empty input:

Step 1: Input x. If x is not an n-query K-machine in standard form then reject.

Step 2: For $i = 1$ to $n+1$ do the following:

- **(a)** Construct a machine x_i that computes as follows: Using the values $\chi_K(x_1), \ldots, \chi_K(x_{i-1})$ computed in step 2(b) as the first $i-1$ oracle answers, simulate x on empty input until x has halted or x is about to make its ith query q_i; if x is about to make its ith query q_i then simulate q_i until q_i has halted.

- **(b)** Ask K whether x_i halts.

(* If x makes an ith query q_i, then $\chi_K(x_i) = \chi_K(q_i)$. *)

Step 3: Output the value of $\chi_K(x_{n+1})$ that was computed in step 2(b).

We assert that x halts if and only if x_{n+1} halts. Let j be the actual number of queries made by machine x. By construction, $x_i \in K$ if and only if $q_i \in K$ for $1 \leq i \leq j$. If x halts then x_{j+1} through x_{n+1} halt; if x diverges then x_{j+1} through x_{n+1} diverge.

The algorithm constructed above makes exactly $n+1$ queries to K. ∎

Theorem 3.2.7 $\mathrm{Q}(n,K) \subset \mathrm{Q}(n+1,K)$.

Proof: The inclusion is obvious (Observation 2.2.2). Lemmas 3.2.5(i) and 3.2.6 imply that

$$\mathrm{H}_{\mathrm{MQ}^*(n,K)} \in \mathrm{Q}(n+1,K) - \mathrm{Q}(n,K).$$

Therefore the inclusion is proper. ∎

Corollary 3.2.8 $\mathrm{FQ}(n,K) \subset \mathrm{FQ}(n+1,K)$.

Proof: This follows from Theorem 3.2.7 and Observation 2.2.3. ∎

Corollary 3.2.9 $\mathrm{Q}_{||}(2^n-1,K) \subset \mathrm{Q}_{||}(2^{n+1}-1,K)$.

Proof: By Theorem 3.2.7, $\mathrm{Q}(n,K) \subset \mathrm{Q}(n+1,K)$. By Theorem 3.2.2, $\mathrm{Q}(n,K) = \mathrm{Q}_{||}(2^n-1,K)$ and $\mathrm{Q}(n+1,K) = \mathrm{Q}_{||}(2^{n+1}-1,K)$. ∎

From Corollary 3.2.9 we can prove that more partial functions are computable with $n+1$ *parallel* queries to K than with only n *parallel* queries to K.

Lemma 3.2.10 $(\forall n)[\mathrm{FQ}_{||}(n,K) \subset \mathrm{FQ}_{||}(n+1,K)]$.

Proof: Proof by contradiction. Assume that $\mathrm{FQ}_{||}(n+1,K) = \mathrm{FQ}_{||}(n,K)$ for some n. Then, by Observation 2.2.15,

$$(\forall m \geq n)[\mathrm{FQ}_{||}(m,K) = \mathrm{FQ}_{||}(n,K)].$$

In particular $\mathrm{FQ}_{||}(2^n-1,K) = \mathrm{FQ}_{||}(2^{n+1}-1,K)$. Therefore, by Observation 2.2.3, $\mathrm{Q}_{||}(2^n-1,K) = \mathrm{Q}_{||}(2^{n+1}-1,K)$, which contradicts Corollary 3.2.9. ∎

Corollary 3.2.11 *There is a total function in* $\mathrm{FQ}_{||}(n+1,K) - \mathrm{FQ}_{||}(n,K)$.

Proof: By Lemma 3.2.10, $\mathrm{FQ}_{||}(n,K) \subset \mathrm{FQ}_{||}(n+1,K)$. Therefore,

$$\mathrm{F}^K_{n+1} \in \mathrm{FQ}_{||}(n+1,K) - \mathrm{FQ}_{||}(n,K).$$

∎

We have shown that $n+1$ parallel queries are more useful than n parallel queries for computing functions. Next we show that $n+1$ parallel queries are more useful than n parallel queries for solving *decision problems*.

Lemma 3.2.12 $\mathrm{PARITY}^K_{n+1} \notin \mathrm{Q}_{||}(n, K)$

Proof: By contradiction. Assume that $\mathrm{PARITY}^K_{n+1} \in \mathrm{Q}_{||}(n, K)$. By Lemma 3.1.8,

$$\begin{aligned}
\#^K_{2n+1} &\in \mathrm{FREC} \circ (\mathrm{FQ}(1, \#^K_n) \parallel \mathrm{FQ}(1, \mathrm{PARITY}^K_{n+1})) \\
&\subseteq \mathrm{FREC} \circ (\mathrm{FQ}(1, \#^K_n) \parallel \mathrm{FQ}_{||}(n, K)) \qquad \text{by assumption} \\
&= \mathrm{FREC} \circ (\mathrm{FQ}_{||}(n, K) \parallel \mathrm{FQ}_{||}(n, K)) \qquad \text{by Lemma 3.1.5(ii)} \\
&= \mathrm{FQ}_{||}(2n, K) \qquad \text{by Corollary 2.2.14.}
\end{aligned}$$

Thus,

$$\#^K_{2n+1} \in \mathrm{FQ}_{||}(2n, K),$$

and so, by Lemma 3.1.5(ii),

$$\mathrm{FQ}_{||}(2n+1, K) \subseteq \mathrm{FQ}(2n, K),$$

which contradicts Lemma 3.2.10. ∎

Theorem 3.2.13 $\mathrm{Q}_{||}(n, K) \subset \mathrm{Q}_{||}(n+1, K)$.

Proof: The containment is obvious. By Lemma 3.2.12,

$$\mathrm{PARITY}^K_{n+1} \in \mathrm{Q}_{||}(n+1, K) - \mathrm{Q}_{||}(n, K),$$

so the containment is proper. ∎

We have shown that the hierarchy of bounded parallel query classes relative to K is proper. We have also seen where the bounded serial query classes fit into the hierarchy:

$$\begin{aligned}
\mathrm{FQ}_{||}(1, K) &= \mathrm{FQ}(1, K) \subset \\
\mathrm{FQ}_{||}(2, K) \subset \mathrm{FQ}_{||}(3, K) &= \mathrm{FQ}(2, K) \subset \mathrm{FQ}_{||}(4, K) \subset \\
\mathrm{FQ}_{||}(5, K) \subset \mathrm{FQ}_{||}(6, K) \subset \mathrm{FQ}_{||}(7, K) &= \mathrm{FQ}(3, K) \subset \mathrm{FQ}_{||}(8, K) \subset \cdots \\
\cdots \subset \mathrm{FQ}_{||}(2^n - 2, K) \subset \mathrm{FQ}_{||}(2^n - 1, K) &= \mathrm{FQ}(n, K) \subset \mathrm{FQ}(2^n, K) \subset \cdots
\end{aligned}$$

The same relationship is true for decision problems:

$$
\begin{array}{rcl}
Q_{||}(1,K) & = & Q(1,K) \subset \\
Q_{||}(2,K) \subset Q_{||}(3,K) & = & Q(2,K) \subset Q_{||}(4,K) \subset \\
Q_{||}(5,K) \subset Q_{||}(6,K) \subset Q_{||}(7,K) & = & Q(3,K) \subset Q_{||}(8,K) \subset \cdots \\
\cdots \subset Q_{||}(2^n-2,K) \subset Q_{||}(2^n-1,K) & = & Q(n,K) \subset Q(2^n,K) \subset \cdots
\end{array}
$$

3.3 A Normal Form for Languages in $Q_{||}(n, K)$

By the definition of $Q_{||}$ every language in $Q_{||}(n,K)$ is weak truth-table reducible to K. In this section, we show that every language in $Q_{||}(n,K)$ is, in fact, truth table reducible to K. Furthermore, the truth table used in the reduction can always be chosen to be n-ary exclusive-or (parity) or its complement.

Theorem 3.3.1 *If* $L \in Q_{||}(n,K)$ *then* $L \leq_{\text{1-tt}} \text{PARITY}_n^K$.

Proof: Let $L \in Q_{||}(n,K)$. Since $Q_{||}(n,K) = Q_{||}(1, \#_n^K)$, let χ_L be computed by a machine M in $MQ_{||}(1, \#_n^K)$. Because M halts on all inputs, we can assume without loss of generality that M always makes exactly one query to $\#_n^K$.

For $1 \leq i \leq n$, we compute a partial function $f_i(x)$ as follows:

Step 1: Simulate M on input x until M prepares its query $\vec{q} = (q_1, \ldots, q_n)$ to $\#_n^K$.

Step 2: Timeshare the following two computations until one of them has terminated:

(a) Simulate $q_1, \ldots, q_n$ until at least i of them have halted, then simulate M assuming that its oracle answer is equal to i, and finally output the value output by M;

(b) Simulate $q_1, \ldots, q_n$ until at least $i+1$ of them have halted, and then output the value 1;

(* If $\#_n^K(\vec{q}\,) = i$, then step 2(a) must terminate because the oracle answer is correct and M halts on all inputs; if $\#_n^K(\vec{q}\,) > i$ then step 2(b) must terminate. If $\#_n^K(\vec{q}\,) < i$ then step 2(a) and step 2(b) both diverge. Thus step 2 terminates, *i.e.*, $f_i(x)$ is defined, if and only if $i \leq \#_n^K(\vec{q}\,)$. *)

Thus,

$$f_i(x) = \begin{cases} 0 \text{ or } 1 \text{ (don't care)} & \text{if } i < \#_n^K(\vec{q}\,) \\ \chi_L(x) & \text{if } i = \#_n^K(\vec{q}\,) \\ \text{diverge} & \text{if } i > \#_n^K(\vec{q}\,). \end{cases}$$

For $0 \leq i \leq n-1$, we define a $\emptyset$-machine $y_i(x)$ that halts if and only if $f_i(x)$ converges, $f_{i+1}(x)$ converges, and $f_i(x) \neq f_{i+1}(x)$. Thus,

$$\chi_K(y_i(x)) = \begin{cases} f_i(x) \oplus f_{i+1}(x) & \text{if } i < \#_n^K(\vec{q}\,) \\ 0 & \text{otherwise.} \end{cases}$$

Let $t = \#_n^K(\vec{q}\,)$. Then

$$\begin{aligned}
&f_0(x) \oplus \mathrm{PARITY}_n^K(y_0(x), \ldots, y_{n-1}(x)) \\
&\quad = f_0(x) \oplus (\chi_K(y_0(x)) \oplus \cdots \oplus \chi_K(y_{n-1}(x))) \\
&\quad = f_0(x) \oplus (\chi_K(y_0(x)) \oplus \cdots \oplus \chi_K(y_{t-1}(x))) \oplus (\chi_K(y_t(x)) \oplus \cdots \oplus \chi_K(y_{n-1}(x))) \\
&\quad = f_0(x) \oplus ((f_0(x) \oplus f_1(x)) \oplus \cdots \oplus (f_{t-1}(x) \oplus f_t(x))) \oplus (0 \oplus \cdots \oplus 0) \\
&\quad = f_0(x) \oplus ((f_0(x) \oplus f_1(x)) \oplus \cdots \oplus (f_{t-1}(x) \oplus f_t(x))) \\
&\quad = ((f_0(x) \oplus f_0(x)) \oplus \cdots \oplus (f_{t-1}(x) \oplus f_{t-1}(x))) \oplus f_t(x) \\
&\quad = f_t(x) \\
&\quad = \chi_L(x).
\end{aligned}$$

Since $f_0(x)$ is total recursive, $L \leq_{1\text{-tt}} \mathrm{PARITY}_n^K$. ∎

Corollary 3.3.2

i. $L \in \mathrm{Q}_{||}(n, K)$ *if and only if* $L \leq_{n\text{-tt}} K$*.*

ii. $L \in \mathrm{Q}(n, K)$ *if and only if* $L \leq_{(2^n-1)\text{-tt}} K$*.*

Proof:

i. The forward implication follows immediately from Theorem 3.3.1. The converse is obvious from the definitions.

ii. This follows from (i), because $\mathrm{Q}(n, K) = \mathrm{Q}_{||}(2^n - 1, K)$ by Theorem 3.2.2.

∎

By definition, $\mathrm{Q}_{||}(n, K)$ consists of exactly those languages that are n-wtt reducible to K. Thus Corollary 3.3.2(i) implies that n-wtt reducibility to K is equivalent to n-tt reducibility to K. As mentioned in the introduction, Lachlan showed in [Lac65] that relative to some oracles n-wtt reducibility need not imply n-tt reducibility or even unbounded tt-reducibility, even when $n = 1$. In [Rog67], it was shown that if $K \leq_{\mathrm{tt}} B$ then

$$(A \leq_{\mathrm{wtt}} B) \text{ implies } (A \leq_{\mathrm{tt}} B);$$

hence, in particular

$$(A \leq_{\mathrm{wtt}} K) \text{ implies } (A \leq_{\mathrm{tt}} K).$$

In Rogers's proof sketch, however, the tt-reduction uses more queries than the wtt-reduction, and thus those methods do not yield our result that n-wtt reducibility to K is equivalent to n-tt reducibility to K.

The next Corollary says that PARITY_n^K is 1-query complete for $\mathrm{Q}_{||}(n, K)$. It is analogous to Lemma 3.1.5(ii).

Corollary 3.3.3 $\mathrm{Q}_{||}(n, K) = \mathrm{Q}(1, \mathrm{PARITY}_n^K)$.

Proof: By Theorem 3.3.1, $\mathrm{Q}_{||}(n, K) \subseteq \mathrm{Q}(1, \mathrm{PARITY}_n^K)$. The reverse containment is obvious. ∎

That tt-reducibility to K via a *fixed* truth table of norm n implies membership in $\mathrm{Q}(1, \mathrm{PARITY}_n^K)$ also follows from [Hay78], where it is shown that $B \leq_{n\text{-tt}} K$ via a reduction whose truth table is independent of the input if and only if

$$(B \leq_1 \mathrm{PARITY}_n^K) \text{ or } (\bar{B} \leq_1 \mathrm{PARITY}_n^K).$$

3.4 Several Rounds of Parallel Queries

In this section, we determine what functions can be computed if we are allowed to make n_1 parallel queries to K, followed by n_2 parallel queries to K, ... followed by n_r parallel queries to K.

Lemma 3.4.1

$$\mathrm{FQ}_{||}((n_1+1)\cdots(n_r+1)-1), K) \subseteq \mathrm{FQ}_{||}(n_r, K) \circ \mathrm{FQ}_{||}(n_{r-1}, K) \circ \cdots \circ \mathrm{FQ}_{||}(n_1, K).$$

Proof: Let $N = (n_1+1)\cdots(n_r+1)-1$. Then

$$\begin{aligned}
\mathrm{FQ}_{||}(N, K) &\subseteq \mathrm{FQ}(1, \#_N^K) \qquad \text{by Lemma 3.1.5(ii)} \\
&\subseteq \mathrm{FQ}_{||}(n_r, \mathrm{GEQ}^K) \circ \mathrm{FQ}_{||}(n_{r-1}, \mathrm{GEQ}^K) \circ \cdots \circ \mathrm{FQ}_{||}(n_1, \mathrm{GEQ}^K) \\
&\quad \text{by Observation 2.2.19} \\
&= \mathrm{FQ}_{||}(n_r, K) \circ \mathrm{FQ}_{||}(n_{r-1}, K) \circ \cdots \circ \mathrm{FQ}_{||}(n_1, K) \\
&\quad \text{because } \mathrm{GEQ}^K \equiv_{\mathrm{m}} K.
\end{aligned}$$

∎

Lemma 3.4.2 $\mathrm{FQ}_{||}(n_2, K) \circ \mathrm{FQ}_{||}(n_1, K) \subseteq \mathrm{FQ}_{||}((n_1+1)(n_2+1)-1, K)$.

Proof: Let

$$f \in \mathrm{FQ}_{||}(n_2, K) \circ \mathrm{FQ}_{||}(n_1, K) = \mathrm{FQ}(1, \#_{n_2}^K) \circ \mathrm{FQ}(1, \#_{n_1}^K),$$

by Lemma 3.1.5(ii). Then $f = f_2 \circ f_1$, where $f_2 \in \mathrm{FQ}(1, \#_{n_2}^K)$ and $f_1 \in \mathrm{FQ}(1, \#_{n_1}^K)$. Let M_2 compute f_2, and let M_1 compute f_1. Without loss of generality, we assume that M_2 makes its query whenever it converges and that M_1 makes its query whenever it converges.

For $1 \leq r \leq n_1$, we define a $\emptyset$-machine x_1^r that computes as follows: simulate M_1 until M_1 prepares its query $(x_1, \ldots, x_{n_1})$; timeshare the $\emptyset$-machines $x_1, \ldots, x_{n_1}$ until at least r of them have halted; and halt.

For $0 \leq s \leq n_1$ and $1 \leq t \leq n_2$, we define a $\emptyset$-machine $x_2^{s,t}$ that computes as follows: simulate M_1 until M_1 prepares its query $(x_1, \ldots, x_{n_1})$; timeshare the $\emptyset$-machines $x_1, \ldots, x_{n_1}$ until at least s of them have halted; complete the simulation of M_1 assuming that the answer to its query is equal to s; simulate M_2, using the output of M_1 as input, until M_2 prepares its query $(y_1, \ldots, y_{n_2})$; timeshare $y_1, \ldots, y_{n_2}$ until at least t of them have halted; and halt.

We simulate $M_2 \circ M_1$ as follows: Ask K in parallel whether x_1^r halts for $1 \leq r \leq n_1$ and whether $x_2^{s,t}$ halts for $0 \leq s \leq n_1$, $1 \leq t \leq n_2$. The answers to those queries to K determine the answer to M_1's query to $\#_{n_1}^K$ and the answer to M_2's query to $\#_{n_2}^K$. We simulate $M_2 \circ M_1$ using this information in lieu of making additional oracle queries. The number of parallel queries used by this simulation is $n_1 + (n_1 + 1)n_2 = (n_1 + 1)(n_2 + 1) - 1$. ∎

Theorem 3.4.3

$$\mathrm{FQ}_{||}(n_2, K) \circ \mathrm{FQ}_{||}(n_1, K) = \mathrm{FQ}_{||}((n_1 + 1)(n_2 + 1) - 1, K).$$

Proof: This follows from Lemma 3.4.1 and Lemma 3.4.2. ∎

Theorem 3.4.4

$$\mathrm{FQ}_{||}(n_r, K) \circ \mathrm{FQ}_{||}(n_{r-1}, K) \circ \cdots \circ \mathrm{FQ}_{||}(n_1, K) = \mathrm{FQ}_{||}((n_1 + 1) \cdots (n_r + 1) - 1, K).$$

Proof: Proof by induction on r. The result is identically true for $r = 1$. Assume the result for some r. We will prove it for $r + 1$.

$$\begin{aligned}
&\mathrm{FQ}_{||}(n_{r+1}, K) \circ \mathrm{FQ}_{||}(n_r, K) \circ \cdots \circ \mathrm{FQ}_{||}(n_1, K) \\
&\quad = \mathrm{FQ}_{||}(n_{r+1}, K) \circ (\mathrm{FQ}_{||}(n_r, K) \circ \cdots \circ \mathrm{FQ}_{||}(n_1, K)) \\
&\quad = \mathrm{FQ}_{||}(n_{r+1}, K) \circ \mathrm{FQ}_{||}((n_1 + 1) \cdots (n_r + 1) - 1, K) \qquad \text{by the induction hypothesis} \\
&\quad = \mathrm{FQ}_{||}(((n_1 + 1) \cdots (n_r + 1) - 1 + 1)(n_{r+1} + 1) - 1, K) \qquad \text{by Theorem 3.4.3} \\
&\quad = \mathrm{FQ}_{||}((n_1 + 1) \cdots (n_{r+1} + 1) - 1, K).
\end{aligned}$$

∎

3.5 The Query Complexity Measure

Definition 3.5.1

- The *query complexity* of a computation relative to B is the number of queries made to B if the computation terminates, infinite otherwise.
- If a computation makes only one round of parallel queries to B then the *parallel-query complexity* of that computation is equal to its *query complexity*.

A measure is a *Blum complexity measure* if it satisfies the following two conditions: (1) the complexity assigned to every divergent computation must be infinite, and (2) there must be an algorithm to determine whether the complexity of a computation is at least c, for finite c (see, for example, [MY78, page 142]). We relativize Blum's definition to apply to computations that use an oracle.[1]

Definition 3.5.2 A measure $\mu(M, x)$ is a *relativized Blum complexity* measure for computations that use oracle B if the measure satisfies the following two conditions:

i. $\mu(M, x) < \infty$ if and only if the B-machine M converges on input x.

[1] Our definition is different from that of Lynch, Meyer, and Fischer [LMF76], because their definition is uniform in the oracle.

ii. The predicate $\mu(M, x) \geq c$ is recursive in B.

Theorem 3.5.3 *Query complexity and parallel-query complexity are relativized Blum complexity measures for computations that use the oracle B if and only if $K \leq_{\mathrm{T}} B$.*

Proof: First, assume that $K \leq_{\mathrm{T}} B$. By definition, the parallel-query complexity and query complexity of a computation are finite if and only if the computation terminates, so condition (i) is satisfied.

We define a "checkpoint" of a computation as a halt or a query. A "segment" is the portion of a computation that occurs between checkpoints. Using an oracle for K, we can determine whether a computation reaches a checkpoint. We can determine if the serial query complexity of a computation is at least c by simulating the computation one segment at a time, as follows:

Step 1: Input a machine M and a string x.

Step 2: For $i = 1$ to c perform the following steps:

(a) Using K, determine whether another checkpoint will be reached. If not (∗ then the computation diverges, so its complexity is infinite, which is at least c ∗) then accept.

(b) (∗ The computation reaches another checkpoint. ∗) Simulate the computation up to the checkpoint. If the checkpoint is a halt (∗ then the computation makes fewer than c queries ∗) then reject.

(c) (∗ The checkpoint is a query. ∗) Using B, answer the query.

Step 3: (∗ If we reach this step then the computation has made c queries. ∗) Accept.

Since $K \leq_{\mathrm{T}} B$, the algorithm above is recursive in B. Since parallel-query complexity is the same as query complexity, whenever the parallel-query complexity is defined, the same algorithm determines if the parallel-query complexity of a computation is at least c. Thus, condition (ii) is also satisfied, so query complexity and parallel-query complexity are relativized Blum computational complexity measures.

Conversely, assume that parallel-query complexity or query complexity is a relativized Blum computational complexity measure. Suppose that we want to know whether $\emptyset$-machine x halts on empty input. We replace x with an equivalent B-machine that never uses its oracle. The query complexity of that B-machine's computation on empty input is either 0 or infinity. By condition (ii), we can determine whether its query complexity is at least 1, via an algorithm that is recursive in B. If the complexity is at least 1 then $x \notin K$; otherwise, $x \in K$. Therefore, $K \leq_{\mathrm{T}} B$. ∎

We will determine the exact query complexity relative to K of several problems.

3.5.1 Halting problems for K-machines

We write K_n to denote the halting problem for n-parallel-query K-machines.

Definition 3.5.4 $K_n = \mathrm{H}_{\mathrm{MQ}^*_{||}(n,K)}$.

Lemma 3.5.5

$$\mathrm{H}_{\mathrm{MQ}^*(n,K)} \equiv_{\mathrm{m}} \mathrm{H}_{\mathrm{MQ}^*_{||}(2^n-1,K)}.$$

Proof: In the proof of Theorem 3.2.2 we showed how to effectively convert a machine in $\mathrm{MQ}^*(n,K)$ into a machine in $\mathrm{MQ}^*(2^n-1,K)$ that computes the same function, and vice versa. If one halts, the other halts. ∎

We shall determine the parallel-query complexity of K_n relative to K. If n is of the form $2^t - 1$, then our previous results provide an upper bound on the complexity of K_n:

$$\begin{aligned}
K_n &= \mathrm{H}_{\mathrm{MQ}^*_{||}(2^t-1,K)} \\
&= \mathrm{H}_{\mathrm{MQ}^*(t,K)} \qquad \text{by Lemma 3.5.5} \\
&\in \mathrm{Q}(t+1,K) \qquad \text{by Lemma 3.2.6} \\
&= \mathrm{Q}(1,K) \circ \mathrm{FQ}(t,K) \\
&= \mathrm{Q}(1,K) \circ \mathrm{FQ}_{||}(2^t-1,K) \qquad \text{by Theorem 3.2.2} \\
&= \mathrm{Q}(1,K) \circ \mathrm{FQ}_{||}(n,K). \\
&= \mathrm{Q}_{||}(2n+1,K) \qquad \text{by Theorem 3.4.3.}
\end{aligned}$$

In the next lemma, we prove this result for all n.

Lemma 3.5.6 $K_n \in \mathrm{Q}_{||}(2n+1, K)$.

Proof: The following algorithm solves the halting problem for n-parallel-query K-machines:

Step 1: Input x; if x is not an n-parallel-query K-machine in standard form then reject.

Step 2: (* Normalize x. *) Let $\hat{x}$ be an n-parallel-query K-machine that computes the same partial function as x, and makes exactly n parallel queries to K whenever it halts.

Step 3: For $1 \le i \le n$, define a $\emptyset$-machine x_i that does the following: simulate $\hat{x}$ until $\hat{x}$ prepares its list of queries $q_1, \ldots, q_n$; then simulate q_i until q_i halts. (* Thus $x_i \in K$ if and only if $\hat{x}$ asks a round of parallel queries and the ith query belongs to K. *)

Compute $\mathrm{F}_n^K(x_1, \ldots, x_n)$.

Step 4: Define a machine x' that computes as follows: simulate $\hat{x}$ until $\hat{x}$ prepares its list of queries; using the oracle answers obtained in step 3, continue the simulation of $\hat{x}$ until $\hat{x}$ halts.

Ask K whether x' halts; output that answer.

We consider two cases.

Case 1: *The machine $\hat{x}$ makes its round of parallel queries.*
In this case, step 3 produces the correct oracle answers. Therefore x' correctly simulates $\hat{x}$, so x' halts if and only if $\hat{x}$ halts.

Case 2: *The machine $\hat{x}$ does not make its round of parallel queries.*
In this case, the machine $\hat{x}$ does not halt (because $\hat{x}$ is in normal form). The machine x' does not halt because x' goes into a divergent computation waiting for $\hat{x}$ to make its round of parallel queries.

In either case x' halts if and only if $\hat{x}$ halts. Because $\hat{x}$ and x compute the same partial function, $\hat{x}$ halts if and only if x halts. Thus x' halts if and only if x halts.

The algorithm above makes only n parallel queries to K followed by a single query to K. Therefore, by Observation 2.2.11,

$$\begin{aligned} K_n &\in \mathrm{FQ}(1,K) \circ \mathrm{FQ}_{||}(n,K) \\ &= \mathrm{FQ}_{||}(2n+1,K) \qquad \text{by Theorem 3.4.3,} \end{aligned}$$

so $K_n \in \mathrm{Q}_{||}(2n+1,K)$ because K_n is a decision problem. ∎

In Theorem 3.5.8 below, we show that the preceding result is tight.

Lemma 3.5.7 $\mathrm{PARITY}^K_{2n+1} \in \mathrm{Q}(1,K_n)$.

Proof: Suppose that we are to determine whether $\vec{x} \in \mathrm{PARITY}^K_{2n+1}(\vec{x}\,)$, where $\vec{x} = (x_1,\ldots,x_{2n+1})$. By Lemma 3.1.2, $K \equiv_{\mathrm{m}} \mathrm{GEQ}^K$. Therefore we can compute $\vec{y} = (y_1,\ldots,y_{2n+1})$ such that $y_i \in K$ if and only if $(i;\vec{x}\,) \in \mathrm{GEQ}^K$. We define a machine M in $\mathrm{MQ}^*_{||}(n,K)$ that does the following: compute $u = \#^K_n(y_2,y_4,\ldots,y_{2n})$; simulate y_{2u+1} until it has halted; and then halt. Then

$$\begin{aligned} \#^K_{2n+1}(\vec{x}\,) &= \#^K_{2n+1}(\vec{y}\,) \qquad \text{by the construction of } \vec{y} \\ &= \begin{cases} 2u & \text{if } y_{2u+1} \notin K \\ 2u+1 & \text{otherwise.} \end{cases} \end{aligned}$$

Thus $\#^K_{2n+1}(\vec{x}\,)$ is odd if and only if y_{2u+1} halts. By construction, M halts if and only if y_{2u+1} halts, so $\mathrm{PARITY}^K_{2n+1}(\vec{x}\,) = 1$ if and only if M halts. Therefore, since M is an n-parallel-query K-machine in standard form,

$$\vec{x} \in \mathrm{PARITY}^K_{2n+1} \Leftrightarrow M \in K_n.$$

Thus

$$\mathrm{PARITY}^K_{2n+1} \leq_{\mathrm{m}} K_n.$$

∎

Theorem 3.5.8 *K_n is 1-query complete for* $\mathrm{Q}_{||}(2n+1, K)$.

Proof: By Lemma 3.5.6, $K_n \in \mathrm{Q}_{||}(2n+1, K)$. Furthermore,

$$\begin{aligned} \mathrm{Q}_{||}(2n+1, K) &\subseteq \mathrm{Q}(1, \mathrm{PARITY}^K_{2n+1}) \qquad \text{by Corollary 3.3.3} \\ &\subseteq \mathrm{Q}(1, K_n) \qquad \text{by Lemma 3.5.7.} \end{aligned}$$

∎

Since K_n is 1-query complete for $\mathrm{Q}_{||}(2n+1, K)$, the parallel-query complexity of K_n relative to K is exactly $2n+1$.

Corollary 3.5.9 $K_n \in \mathrm{Q}_{||}(2n+1, K) - \mathrm{Q}_{||}(2n, K)$.

Proof: By Theorem 3.5.8, $K_n \in \mathrm{Q}_{||}(2n+1, K)$ and $\mathrm{Q}_{||}(2n+1, K) \subseteq \mathrm{Q}(1, K_n)$. If $K_n \in \mathrm{Q}_{||}(2n, K)$ then $\mathrm{Q}_{||}(2n+1, K) \subseteq \mathrm{Q}_{||}(2n, K)$, contradicting Theorem 3.2.13; therefore $K_n \notin \mathrm{Q}_{||}(2n, K)$. ∎

Having seen that $\mathrm{Q}_{||}(2n+1, K) \subseteq \mathrm{Q}(1, K_n)$, one is led to wonder if a similar result is true for functions: Is it possible that $\mathrm{FQ}_{||}(2n+1, K) \subseteq \mathrm{FQ}(1, K_n)$? The next theorem rules out that possibility.

Theorem 3.5.10 *If B is any set in* $\mathrm{Q}_{||}(n, K)$, *then* $\mathrm{FQ}_{||}(2, K) \not\subseteq \mathrm{FQ}(1, B)$.

Proof: By contradiction. Let $B \in \mathrm{Q}_{||}(n, K)$ and suppose that $\mathrm{FQ}_{||}(2, K)$ is a subset of $\mathrm{FQ}(1, B)$. By Theorem 3.4.3,

$$\begin{aligned} \mathrm{FQ}_{||}(3n-1, K) &= \mathrm{FQ}_{||}(n-1, K) \circ \mathrm{FQ}_{||}(2, K) \\ &\subseteq \mathrm{FQ}_{||}(n-1, K) \circ \mathrm{FQ}(1, B). \end{aligned}$$

Thus every function in $\mathrm{FQ}_{||}(3n-1, K)$ is computable by making a machine M that makes one query to B followed by $n-1$ parallel queries to K. We would like to simulate M by making n parallel queries to K in order to determine the result of B, simultaneous with $n-1$ parallel queries to K to determine the result of the entire computation assuming that B answers no, and simultaneous with $n-1$ parallel queries to K to determine the result assuming that B answers yes. However, it is possible that the computation diverges when we assume that B gives the incorrect oracle answer. As usual, we exploit our oracle K in order to prevent our computations from diverging. Without loss of generality we can assume that M makes exactly one query to B followed by exactly $n-1$ parallel queries to K whenever M converges. We simulate M as follows:

Step 1: Simulate M until M is about to make a query to B.

Step 2: For $t = 0, 1$ and $1 \le i \le n-1$, define a machine x_i^t that does the following: simulate M, assuming that the answer to the query to B is t, until M is about to make its parallel queries $q_1, \ldots, q_{n-1}$ to K; simulate q_i until q_i terminates.

Step 3: Perform the following two computations simultaneously:

(a) Make n parallel queries to K in order to evaluate the query to B.

(b) Compute $\mathrm{F}^K_{2n-2}(x_1^0, \ldots, x_{n-1}^0, x_1^1, \ldots, x_{n-1}^1)$.

Step 4: Let t equal the answer to the query to B, as computed in step 3(a); simulate M assuming that the answer to the query to B is t and that the answers to the parallel queries to K are equal to $\mathrm{F}^K_{n-1}(x_1^t, \ldots, x_{n-1}^t)$, as computed in step 3(b).

This algorithm simulates M by making only $3n-2$ parallel queries to K. Therefore,

$$\mathrm{FQ}_{||}(3n-1, K) \subseteq \mathrm{FQ}_{||}(3n-2, K),$$

contradicting Theorem 3.2.13. ∎

Theorem 3.5.11 *If B is any set, then* $\mathrm{FQ}_{||}(2, K) \not\subseteq \mathrm{FQ}(1, B)$.

Proof: By contradiction. Assume that $\mathrm{F}^K_2 \in \mathrm{FQ}(1, B)$, for some B. Let M be a 1-query B-machine that computes F^K_2. Without loss of generality, we can assume that $M^{()}$ never makes more than one query to its oracle, even if the oracle is different from B. For $t = 0, 1$ we define a $\emptyset$-machine M^t that simulates M assuming that the oracle answer is equal to t. Then for all pairs (x, y), one of M^0 or M^1 produces the correct output $\mathrm{F}^K_2(x, y)$.

We define a set A computed as follows: On input (x,y), compute $\mathrm{F}_2^K(x,y)$; time-share M^0 and M^1 on input (x,y) until one of them produces the correct answer $\mathrm{F}_2^K(x,y)$; if the first one to produce the correct answer is M^0 then return 0, otherwise return 1.

Then M^A computes F_2^K by making one query to A, which is in $\mathrm{Q}_{||}(2,K)$. That contradicts Theorem 3.5.10. ∎

In Chapter 4 we will show that if A is *any nonrecursive set* then F_2^A is not in $\mathrm{FQ}(1,B)$ for any set B.

3.5.2 Recursively Defined Halting Problems

We consider one class of decision problems, and we analyze their parallel-query complexity relative to K.

Definition 3.5.12 We define K^n recursively:

$$K^n = \begin{cases} K & \text{if } n = 0 \\ \mathrm{H}_{\mathrm{MQ}^*(1,K^{n-1})} & \text{otherwise.} \end{cases}$$

Thus K^n is the halting problem for 1-query K^{n-1}-machines.[2]

Theorem 3.5.13

$$\mathrm{H}_{\mathrm{MQ}^*(1,\mathrm{PARITY}_n^K)} \in \mathrm{Q}_{||}(n+2,K).$$

Proof: The following algorithm solves the halting problem for 1-query PARITY_n^K-machines:

Step 1: Input M. If M is not a 1-query PARITY_n^K-machine in standard form then reject.

[2]Do not confuse K^n with $K^{(n)}$, which is the nth jump of K, as defined in [Rog67, p. 256] and in [Soa87, Definition 2.2]. We will show that $K^n \leq_{\mathrm{T}} K$, so the Turing degree of K^n is much lower than the Turing degree of $K^{(n)}$. The set K^n is a sort of 1-query jump of K^{n-1}, because $K^n \notin \mathrm{Q}(1,K^{n-1})$ by Lemma 3.2.5(i).

Step 2: For $1 \le i \le n$, define a $\emptyset$-machine x_i that does the following: simulate M until M is about to make its query $q_1, \ldots, q_n$ to PARITY_n^K; simulate q_i until q_i halts;

Step 3: For $t = 0, 1$, define a $\emptyset$-machine M^t that does the following: simulate M assuming that the answer to M's query to PARITY_n^K is equal to t.

Step 4: Compute $\mathrm{F}_{n+2}^K(x_1, \ldots, x_n, M^0, M^1)$.

Step 5: Use the oracle answers obtained in step 4 in order to compute

$$t = \mathrm{PARITY}_n^K(x_1, \ldots, x_n)$$

without making any more queries. (* If M queries its oracle, then the construction guarantees that t is equal to the answer given by the oracle. *)

Step 6: Output the value $\chi_K(M^t)$, which was computed in step 4.

The algorithm given above makes only $n + 2$ parallel queries to K. ∎

Corollary 3.5.14 *If* $B \in \mathrm{Q}_{||}(n, K)$ *then*

$$\mathrm{H}_{\mathrm{MQ}^*(1,B)} \in \mathrm{Q}_{||}(n+2, K).$$

Proof: Let B be a set in $\mathrm{Q}_{||}(n, K)$. Then $B \in \mathrm{Q}(1, \mathrm{PARITY}_n^K)$, by Corollary 3.3.3. Therefore, $\mathrm{FQ}(1, B) \subseteq \mathrm{FQ}(1, \mathrm{PARITY}_n^K)$ by Observation 2.2.5. Furthermore, the proof of Observation 2.2.5 is constructive; it allows us to transform a machine in $\mathrm{MQ}^*(1, B)$ into an equivalent machine in $\mathrm{MQ}^*(1, \mathrm{PARITY}_n^K)$. Therefore,

$$\mathrm{H}_{\mathrm{MQ}^*(1,B)} \le_{\mathrm{m}} \mathrm{H}_{\mathrm{MQ}^*(1,\mathrm{PARITY}_n^K)}.$$

Therefore, by Theorem 3.5.13,

$$\mathrm{H}_{\mathrm{MQ}^*(1,B)} \in \mathrm{Q}_{||}(n+2, K).$$

∎

Lemma 3.5.15 $K^n \in \mathrm{Q}_{||}(2n+1, K)$.

Proof: By induction on n. $K^0 = K \in \mathrm{Q}_{||}(1, K)$, so the base case is established. Assume that $K^n \in \mathrm{Q}_{||}(2n+1, K)$, for some $n \geq 0$. Therefore,

$$\mathrm{H}_{\mathrm{MQ}^*(1,K^n)} \in \mathrm{Q}_{||}(2n+3, K),$$

by Corollary 3.5.14. Therefore $K^{n+1} \in \mathrm{Q}_{||}(2n+3, K)$ by the definition of K^{n+1}, completing the induction. ▮

Lemma 3.5.16

$$\mathrm{PARITY}^K_{2n+1} \leq_{\mathrm{m}} \mathrm{H}_{\mathrm{MQ}^*(1,\mathrm{PARITY}^K_{2n-1})}.$$

Proof: Suppose that we are to determine whether $\vec{x} \in \mathrm{PARITY}^K_{2n+1}$, where $\vec{x} = (x_1, \ldots, x_{2n+1})$. By Lemma 3.1.2, $\mathrm{GEQ}^K \equiv_{\mathrm{m}} K$. Therefore we can compute $\vec{y} = (y_1, \ldots, y_{2n+1})$ such that $y_i \in K$ if and only if $(i; \vec{x}\,) \in \mathrm{GEQ}^K$. We define a 1-query PARITY^K_{2n-1}-machine M that does the following:

Step 1: Simulate y_1 until y_1 has halted. (* If $\#^K_{2n+1}(\vec{x}\,) = 0$, which is even, then this step diverges. *)

Step 2: Let $p = \mathrm{PARITY}^K_{2n-1}(y_2, y_3, \ldots, y_{2n})$. (* Let $t = \#^K_{2n-1}(y_2, y_3, \ldots, y_{2n})$. *)

Step 3: If $p = 0$ then halt. (* In this case t is even. $\#^K_{2n+1}(\vec{x}\,) = t+1$, which is odd. *)

Step 4: If $p = 1$ then simulate y_{2n+1} until y_{2n+1} has halted. (* In this case t is odd. If $t < 2n-1$ then $\#^K_{2n+1}(\vec{x}\,) = t+1$, which is even. If $t = 2n-1$ then $\#^K_{2n+1}(\vec{x}\,) = 2n$ or $2n+1$, depending on whether y_{2n+1} halts. We halt only in the latter case. *)

We convert M to standard form. The 1-query PARITY^K_{2n-1}-machine M halts if and only if $\vec{x} \in \mathrm{PARITY}^K_{2n+1}$. ▮

Corollary 3.5.17 $\mathrm{Q}_{||}(2n+1, K) \subseteq \mathrm{Q}(1, K^n)$.

Proof: By induction on n. Since $\mathrm{Q}_{||}(1, K) = \mathrm{Q}(1, K)$, the base case ($n = 0$) is established. Assume that the corollary is true for some $n-1$, where $n \geq 1$. Then $\mathrm{Q}_{||}(2n-1, K) \subseteq \mathrm{Q}(1, K^{n-1})$, so $\mathrm{PARITY}^K_{2n-1} \in \mathrm{Q}(1, K^{n-1})$. Therefore we can effectively transform any machine in $\mathrm{MQ}^*(1, \mathrm{PARITY}^K_{2n-1})$ into an equivalent machine in $\mathrm{MQ}^*(1, K^{n-1})$, so

$$\mathrm{H}_{\mathrm{MQ}^*(1,\mathrm{PARITY}^K_{2n-1})} \leq_{\mathrm{m}} \mathrm{H}_{\mathrm{MQ}^*(1,K^{n-1})} = K^n.$$

By Lemma 3.5.16,

$$\mathrm{PARITY}^K_{2n+1} \leq_{\mathrm{m}} \mathrm{H}_{\mathrm{MQ}^*(1,\mathrm{PARITY}^K_{2n-1})}.$$

By transitivity,

$$\mathrm{PARITY}^K_{2n+1} \leq_{\mathrm{m}} K^n. \tag{1}$$

By Corollary 3.3.3,

$$\begin{aligned} \mathrm{Q}_{||}(2n+1, K) &\subseteq \mathrm{Q}(1, \mathrm{PARITY}^K_{2n+1}) \\ &\subseteq \mathrm{Q}(1, K^n) \qquad \text{by (1).} \end{aligned}$$

∎

Theorem 3.5.18 *K^n is 1-query complete for $\mathrm{Q}_{||}(2n+1, K)$.*

Proof: This follows from Lemma 3.5.15 and Corollary 3.5.17. ∎

Theorem 3.5.19 *If n is an odd number then $\mathrm{H}_{\mathrm{MQ}^*(1,\mathrm{PARITY}^K_n)}$ is 1-query complete for $\mathrm{Q}_{||}(n+2, K)$.*

Proof: By Theorem 3.5.13,

$$\mathrm{H}_{\mathrm{MQ}^*(1,\mathrm{PARITY}^K_n)} \in \mathrm{Q}_{||}(n+2, K).$$

Because n is odd,

$$\mathrm{PARITY}^K_{n+2} \leq_{\mathrm{m}} \mathrm{H}_{\mathrm{MQ}^*(1,\mathrm{PARITY}^K_n)}.$$

by Lemma 3.5.16. Because PARITY^K_{n+2} is 1-query complete for $\mathrm{Q}_{||}(n+2, K)$,

$$\mathrm{Q}_{||}(n+2, K) \subseteq \mathrm{Q}(1, \mathrm{H}_{\mathrm{MQ}^*(1,\mathrm{PARITY}^K_n)}).$$

∎

By Theorem 3.5.19, the parallel-query-complexity of the halting problem for 1-query PARITY_n^K-machines is $n+2$, for odd n. We do not know the parallel-query-complexity of the halting problem for 1-query PARITY_n^K-machines for even n.

3.6 Unbounded Queries

In Section 3.2, we considered languages that are n-parallel-query reducible to K, and we showed that they form a hierarchy between the recursive languages and the languages that are weak truth-table reducible to K. In this section, we extend the hierarchy up through languages that are Turing reducible to K.

We consider reducibilities that allow an unbounded number of queries. If we allow an unbounded number of serial queries, then the reducibility is the same as Turing reducibility. If, however, we allow only a bounded number of rounds of parallel queries (with an unbounded number of parallel queries allowed during some rounds), then we obtain reducibilities that are different from all of the reducibilities mentioned earlier in this dissertation. If r and s are two reducibilities then we say that r-reducibility is weaker [Odi81] than s-reducibility if for all sets A and B

$$A \leq_s B \Rightarrow A \leq_r B.$$

In this section, we define reducibilities that are weaker than weak truth-table reducibility, but stronger than Turing reducibility. Our goal is a generalization of Theorem 3.4.4 in which an unbounded number of parallel queries may be allowed during some rounds.

We define F_ω^B to be a function that can compute F_n^B for arbitrary n.

Definition 3.6.1 For every n,

$$\mathrm{F}_\omega^B(x_1, \ldots, x_n) = \mathrm{F}_n^B(x_1, \ldots, x_n).$$

We generalize the definition of bounded query classes relative to the oracle B:

Definition 3.6.2

- $\mathrm{MQ}_{||}(\omega, B) = \mathrm{MQ}(1, \mathrm{F}_{\omega}^{B})$.
- $\mathrm{FQ}_{||}(\omega, B)$ is the set of partial functions that are computable by a machine in $\mathrm{MQ}_{||}(\omega, B)$.
- $\mathrm{Q}_{||}(\omega, B)$ is the set of 0,1-valued total functions that are in $\mathrm{FQ}_{||}(\omega, B)$.

Thus, $\mathrm{MQ}_{||}(\omega, B)$ is the set of machines with oracle B that make at most one round of parallel queries to B (with no bound on the number of queries made in the round). $\mathrm{Q}_{||}(\omega, B)$ is the set of languages that are wtt-reducible to B.

We define $\mathrm{PARITY}_{\omega}^{B}$ to be an oracle that can compute PARITY_{n}^{B} for arbitrary n.

Definition 3.6.3 For every n,

$$\mathrm{PARITY}_{\omega}^{B}(x_1, \ldots, x_n) = \mathrm{PARITY}_{n}^{B}(x_1, \ldots, x_n).$$

Theorem 3.6.4 *$B \in \mathrm{Q}_{||}(\omega, K)$ if and only if $B \leq_{\text{1-tt}} \mathrm{PARITY}_{\omega}^{K}$.*

Proof: If $B \leq_{\text{1-tt}} \mathrm{PARITY}_{\omega}^{K}$ then $B \in \mathrm{Q}_{||}(\omega, K)$ by the definition of $\mathrm{Q}_{||}(\omega, K)$. Conversely, let B be any set in $\mathrm{Q}_{||}(\omega, K) = \mathrm{Q}(1, \mathrm{F}_{\omega}^{K})$. Let B be accepted by a machine M belonging to $\mathrm{MQ}(1, \mathrm{F}_{\omega}^{K})$. The following algorithm decides membership in B:

Step 1: Input x.

Step 2: Simulate M on input x until M prepares its query $(q_1, \ldots, q_n)$ to F_{ω}^{K}. (* We only want to figure out n. *)

Step 3: Transform M into a machine that queries F_{n}^{K} instead of querying F_{ω}^{K}.

Step 4: As in the proof of Theorem 3.3.1, transform the n-parallel-query K-machine from step 3 into a machine that computes via a 1-truth-table reduction to PARITY_{n}^{K}.

Step 5: Output the result of running the machine produced by step 4 on input x.

The computations in steps 1 through 4 are total recursive; step 5 computes a 1-truth-table reduction to $\mathrm{PARITY}_{\omega}^{K}$. Thus, $B \leq_{\text{1-tt}} \mathrm{PARITY}_{\omega}^{K}$. ∎

The next theorem states that every partial function computable by a machine that makes an unbounded number of parallel queries to K followed by one query to K is also computable by a machine that makes only one round of parallel queries to K.

Theorem 3.6.5 $\mathrm{FQ}(1, K) \circ \mathrm{FQ}_{||}(\omega, K) = \mathrm{FQ}_{||}(\omega, K)$.

Proof: Let $f \in \mathrm{FQ}(1, K) \circ \mathrm{FQ}_{||}(\omega, K)$. Then $f = f_2 \circ f_1$ where f_2 is computed by a machine M_2 in $\mathrm{FQ}(1, K)$ and f_1 is computed by a machine M_1 in $\mathrm{FQ}(1, \mathrm{F}_\omega^K)$. The following algorithm computes f:

Step 1: Input x.

Step 2: Simulate M_1 on input x until M_1 prepares its query $(q_1, \ldots, q_n)$ to F_ω^K.

Step 3: Transform M_1 into a machine M_1' that queries F_n^K instead of querying F_ω^K. (* We only want to figure out n. *)

Step 4: Using the method of Lemma 3.4.2 transform $M_2 \circ M_1'$ into an equivalent $(2n+1)$-parallel-query K-machine.

Step 5: Output the result of running the machine produced by step 4 on input x.

The computations in steps 1 through 4 are total recursive. Step 5 makes only one round of parallel queries to K. Therefore $f \in \mathrm{FQ}_{||}(\omega, K)$.

The reverse containment is obvious. ∎

Corollary 3.6.6 $\mathrm{FQ}_{||}(n, K) \circ \mathrm{FQ}_{||}(\omega, K) = \mathrm{FQ}_{||}(\omega, K)$.

Proof:

$$\begin{aligned}
\mathrm{FQ}_{||}(n, K) \circ \mathrm{FQ}_{||}(\omega, K) &\subseteq \mathrm{FQ}(n, K) \circ \mathrm{FQ}_{||}(\omega, K) \\
&\subseteq \underbrace{\mathrm{FQ}(1, K) \circ \cdots \circ \mathrm{FQ}(1, K)}_{n} \circ \mathrm{FQ}_{||}(\omega, K) \\
&= \mathrm{FQ}_{||}(\omega, K),
\end{aligned}$$

by repeated application of Theorem 3.6.5 and the associativity of composition. ∎

We adopt the following notational convenience:

Definition 3.6.7

$$\mathrm{Q}_{||}([n]\omega^m, A) = \mathrm{SREC} \circ \underbrace{\mathrm{FQ}_{||}(\omega, A) \circ \cdots \circ \mathrm{FQ}_{||}(\omega, A)}_{m} \circ \mathrm{FQ}_{||}(n, A).$$

Thus $\mathrm{Q}_{||}([n]\omega^m, A)$ is the set of decision problems that can be solved by an algorithm that makes n parallel queries to A followed by m rounds of (unbounded) parallel queries to A. This convention is motivated by Theorem 3.4.3, which gives an approximately multiplicative rule for combining several rounds of parallel queries to K into a single round of parallel queries. The analogy between this convention and ordinal notation will be explained in Section 3.8.

We show that the halting problem for $\mathrm{MQ}^*_{||}([n]\omega^m, K)$ is in $\mathrm{Q}_{||}([2n+1]\omega^m, K)$.

Theorem 3.6.8

$$\mathrm{H}_{\mathrm{MQ}^*_{||}([n]\omega^m, K)} \in \mathrm{Q}_{||}([2n+1]\omega^m, K).$$

Proof: By induction on m. By Lemma 3.2.6, $\mathrm{H}_{\mathrm{MQ}^*_{||}(n,K)} \in \mathrm{Q}_{||}(2n+1, K)$, establishing the base case ($m = 0$). Assume that

$$\mathrm{H}_{\mathrm{MQ}^*_{||}([n]\omega^m, K)} \in \mathrm{Q}_{||}([2n+1]\omega^m, K),$$

for some $m \geq 0$. The following algorithm solves the halting problem for $\mathrm{MQ}^*_{||}([n]\omega^{m+1}, K)$:

Step 1: Input M, a machine in $\mathrm{MQ}_{||}([n]\omega^{m+1})$. Check whether M is in standard form; if not then reject. Transform M into a normal form that makes exactly n parallel queries to K, followed by exactly $m+1$ rounds of parallel queries to K, whenever M halts;

Step 2: Define an n-parallel-query K-machine M' that simulates M until M prepares its *second* round of parallel queries.

Step 3: Determine whether M' halts; if not, then reject. (* By Lemma 3.2.6, we can solve the halting problem for n-parallel-query K-machines by making only $2n+1$ parallel queries to K. *)

Step 4: Simulate M until M prepares its second round of parallel queries $q_1, \ldots, q_j$ to K.

Step 5: Construct a machine $\hat{M}$ by modifying M so that it queries F_j^K instead of F_ω^K on the second round of queries. (* $\hat{M}$ makes n parallel queries to K, followed by j parallel queries to K, followed by m rounds of parallel queries to K. *)

Step 6: Using the technique of Lemma 3.4.2, transform $\hat{M}$ into an equivalent machine in $\mathrm{MQ}_{||}^*([((n+1)(j+1)-1)]\omega^m, K)$.

Step 7: Determine whether the machine constructed in step 6 halts; if so accept, otherwise reject. (* By the induction hypothesis, the halting problem for $\mathrm{MQ}_{||}^*([((n+1)(j+1)-1)]\omega^m, K)$ is in $\mathrm{Q}_{||}([2(n+1)(j+1)-1]\omega^m, K)$, which is a subset of $\mathrm{Q}_{||}(\omega^{m+1}, K)$. *)

This algorithm makes $2n+1$ parallel queries to K during step 3, and $m+1$ rounds of parallel queries to K during step 7. Thus the halting problem for $\mathrm{MQ}_{||}^*([n]\omega^m, K)$ is in $\mathrm{Q}_{||}([2n+1]\omega^m, K)$. ∎

By the preceding theorem, the halting problem for ω-parallel-query K-machines is in $\mathrm{Q}_{||}(\omega, K) \circ \mathrm{FQ}(1, K)$. This contrasts with the proof of Lemma 3.5.6, in which we showed that the halting problem for n-parallel-query K-machines is in $\mathrm{Q}(1, K) \circ \mathrm{FQ}_{||}(n, K)$. Of course, the halting problem for n-parallel-query K-machines is in $\mathrm{Q}_{||}(n, K) \circ \mathrm{FQ}(1, K)$, because

$$\mathrm{Q}_{||}(n, K) \circ \mathrm{FQ}(1, K) = \mathrm{Q}_{||}(2n+1, K) = \mathrm{Q}(1, K) \circ \mathrm{FQ}_{||}(n, K)$$

by Theorem 3.4.3; however, we did not obtain that result directly.

Corollary 3.6.9 $\mathrm{Q}_{||}([n]\omega^m, K) \subset \mathrm{Q}_{||}([2n+1]\omega^m, K)$

Proof: By Theorem 3.6.8, the halting problem for $\mathrm{MQ}_{||}^*([n]\omega^m, K)$ is in $\mathrm{Q}_{||}([2n+1]\omega^m, K)$. By an argument similar to either proof of Lemma 3.2.5, the halting problem for $\mathrm{MQ}_{||}^*([n]\omega^m, K)$ is not in $\mathrm{Q}_{||}([n]\omega^m, K)$. Therefore, $\mathrm{Q}_{||}([n]\omega^m, K)$ is a proper subset of $\mathrm{Q}_{||}([2n+1]\omega^m, K)$. ∎

Theorem 3.6.10 $\mathrm{Q}([n]\omega^m, K) \subset \mathrm{Q}([(n+1)]\omega^m, K)$

Proof: The containment is obvious. We show that the containment is proper by contradiction. Assume that

$$\mathrm{Q}_{||}([n+1]\omega^m, K) \subseteq \mathrm{Q}_{||}([n]\omega^m, K).$$

Equivalently,

$$\mathrm{Q}_{||}(\omega^m, K) \circ \mathrm{FQ}_{||}(n+1, K) \subseteq \mathrm{Q}_{||}(\omega^m, K) \circ \mathrm{FQ}_{||}(n, K).$$

Therefore,

$$(\forall j)[\mathrm{Q}_{||}(\omega^m, K) \circ \mathrm{FQ}_{||}(n+1, K) \circ \mathrm{FQ}_{||}(j, K) \subseteq \mathrm{Q}_{||}(\omega^m, K) \circ \mathrm{FQ}_{||}(n, K) \circ \mathrm{FQ}_{||}(j, K)],$$

so, by Theorem 3.4.3,

$$(\forall j)[\mathrm{Q}_{||}(\omega^m, K) \circ \mathrm{FQ}_{||}((j+1)(n+2)-1, K) \subseteq \mathrm{Q}_{||}(\omega^m, K) \circ \mathrm{FQ}_{||}((j+1)(n+1)-1, K)],$$

and so, by definition,

$$(\forall j)[\mathrm{Q}_{||}([(j+1)(n+2)-1]\omega^m, K) \subseteq \mathrm{Q}_{||}([(j+1)(n+1)-1]\omega^m, K)]. \qquad (2)$$

If $j \geq n$, then

$$(j+2)(n+1)-1 \leq (j+1)(n+2)-1,$$

so

$$(\forall j \geq n)[\mathrm{FQ}_{||}((j+2)(n+1)-1, K) \subseteq \mathrm{FQ}_{||}((j+1)(n+2)-1, K)].$$

Therefore,

$$(\forall j \geq n)[\mathrm{Q}_{||}(\omega^m, K) \circ \mathrm{FQ}_{||}((j+2)(n+1)-1, K) \subseteq \mathrm{Q}_{||}(\omega^m, K) \circ \mathrm{FQ}_{||}((j+1)(n+2)-1, K)],$$

so, by definition,

$$(\forall j \geq n)[\mathrm{Q}_{||}([(j+2)(n+1)-1]\omega^m, K) \subseteq \mathrm{Q}_{||}([(j+1)(n+2)-1]\omega^m, K)]. \qquad (3)$$

By transitivity, equation (3) and equation (2) imply that

$$(\forall j \geq n)[\mathrm{Q}_{||}([(j+2)(n+1)-1]\omega^m, K) \subseteq \mathrm{Q}_{||}([(j+1)(n+1)-1]\omega^m, K)].$$

That statement is true for $j = n, n+1, \ldots, 2n$. Therefore, by transitivity,

$$\mathrm{Q}_{||}([(2n+2)(n+1)-1]\omega^m, K) \subseteq \mathrm{Q}_{||}([(n+1)(n+1)-1]\omega^m, K),$$

so

$$\mathrm{Q}_{||}([2(n+1)^2-1]\omega^m, K) \subseteq \mathrm{Q}_{||}([(n+1)^2-1]\omega^m, K).$$

That contradicts Corollary 3.6.9. ∎

Theorem 3.6.11 $\mathrm{Q}_{||}([n]\omega^m, K) \subset \mathrm{Q}_{||}(\omega^{m+1}, K)$.

Proof:

$$\begin{aligned} \mathrm{Q}_{||}([n]\omega^m, K) &\subset \mathrm{Q}_{||}([n+1]\omega^m, K) \qquad \text{by Theorem 3.6.10} \\ &\subseteq \mathrm{Q}_{||}(\omega^{m+1}, K) \qquad \text{by definition.} \end{aligned}$$

∎

We adopt a further notational convenience:

Definition 3.6.12 $\mathrm{Q}_{||}(2^\omega, A)$ is the set of decision problems that can be solved by making an unbounded number of serial queries to A.

Thus $\mathrm{Q}_{||}(2^\omega, A)$, which also might reasonably be called $\mathrm{Q}(\omega, A)$, is the set of decision problems that are Turing reducible to A.

Theorem 3.6.13 $\mathrm{Q}_{||}([n]\omega^m, K) \subset \mathrm{Q}_{||}(2^\omega, K)$.

Proof:

$$\begin{aligned} \mathrm{Q}_{||}([n]\omega^m, K) &\subset \mathrm{Q}_{||}([n+1]\omega^m, K) \qquad \text{by Theorem 3.6.10} \\ &\subseteq \mathrm{Q}_{||}(2^\omega, K) \qquad \text{by definition.} \end{aligned}$$

∎

This completes our hierarchy of decision problems reducible to K:

$$
\begin{aligned}
&\mathrm{Q}_{||}(1,K) \subset \mathrm{Q}_{||}(2,K) \subset \cdots \subset \mathrm{Q}_{||}(n,K) \subset \mathrm{Q}_{||}(n+1,K) \subset \cdots \\
&\quad \subset \mathrm{Q}_{||}(\omega,K) \subset \mathrm{Q}_{||}([1]\omega,K) \subset \cdots \subset \mathrm{Q}_{||}([n]\omega,K) \subset \mathrm{Q}_{||}([n+1]\omega,K) \subset \cdots \\
&\quad \vdots \\
&\quad \subset \mathrm{Q}_{||}(\omega^m,K) \subset \mathrm{Q}_{||}([1]\omega^m,K) \subset \cdots \subset \mathrm{Q}_{||}([n]\omega^m,K) \subset \mathrm{Q}_{||}([n+1]\omega^m,K) \subset \cdots \\
&\quad \subset \mathrm{Q}_{||}(\omega^{m+1},K) \subset \mathrm{Q}_{||}([1]\omega^{m+1},K) \subset \cdots \subset \mathrm{Q}_{||}([n]\omega^{m+1},K) \subset \mathrm{Q}_{||}([n+1]\omega^{m+1},K) \subset \cdots \\
&\quad \vdots \\
&\quad \subset \mathrm{Q}_{||}(2^\omega,K).
\end{aligned}
$$

3.7 Chromatic Number of a Recursive Graph

A graph $G = (V, E)$ is said to be recursive if its vertex set V and its edge set E are countable and recursive. The chromatic number of a graph is the minimum number of colors that suffice to color the graph in such a way that no two adjacent vertices have the same color. If we know that a recursive graph can be colored with a finite number of colors, then we can compute its chromatic number with a K-machine. In fact, given an *a priori* bound c on the chromatic number of G, the chromatic number of G can be computed by making $\lceil \log(c+1) \rceil$ serial queries to K; this result is tight [BG89].

In this section, we consider the problem of computing the chromatic number, $\chi(G)$, of a graph when we are not given an *a priori* bound on its chromatic number. We find tight bounds on the query complexity of computing the chromatic number of a graph; we express the complexity as a function of the chromatic number.

Computing the chromatic number of a recursive graph is a special case of a more general problem called unbounded searching, which was posed by Bentley and Yao in [BY76]. The problem is as follows: Player A chooses an arbitrary natural number, n. Player B is allowed to ask whether a natural number x is less than n. In general the number of questions that B has to ask in order to determine n is a function f of n. How small can this function be?

Theorem 3.7.1 *If there is an unbounded searching algorithm that asks only $f(n)$ questions to determine the number n, then there is an algorithm that computes $\chi(G)$ for recursive graphs by making only $f(\chi(G))$ serial queries to K.*

Proof: The chromatic number of a graph is some natural number n. In [BG89], it was shown that we can determine if $n > t$ by making one query to K. Thus we can determine n by using unbounded search; the number of queries we make to K is equal to the number of questions asked in the unbounded search. ∎

In [Bei90], we showed that for any $\epsilon > 0$

$$f(n) = \left(\sum_{1 \leq i \leq \log^* n} \log^{(i)}(n) \right) - (\log\log(e - \epsilon)) \log^* n + O(1) \qquad (4)$$

questions are sufficient, but

$$f(n) = \left(\sum_{1 \leq i \leq \log^* n} \log^{(i)}(n) \right) - (\log\log e) \log^* n + O(1)$$

questions are not sufficient. Those bounds are slightly tighter than the original bounds provided by Bentley and Yao. We also proved the existence of algorithms that differ from optimal by an arbitrarily small total recursive function. Very tight bounds were provided constructively by Knuth in [Knu81].

Theorem 3.7.2 *Let ϵ be any positive real number. There is an algorithm that computes $\chi(G)$ for recursive graphs by making only*

$$f(n) = \left(\sum_{1 \leq i \leq \log^* n} \log^{(i)}(n) \right) - (\log\log(e - \epsilon)) \log^* n + O(1)$$

serial queries to K.

Proof: By equation (4), there is an unbounded searching algorithm that asks only $f(n)$ questions to determine the number n. Therefore, by Theorem 3.7.1, there is an algorithm that computes $\chi(G)$ for recursive graphs by making only $f(n)$ serial queries to K. ∎

In [Bei90] we proved the following result:

Theorem 3.7.3 *Let f be a nondecreasing, total recursive function such that*

$$s = \sum_{i \geq 1} 2^{-f(i)}$$

is a recursive real number (i.e., there is an algorithm that computes each bit of a binary expansion of s) and $s \leq 1$. Then there is an algorithm that solves the unbounded searching problem by asking at most $f(n)$ questions.

Corollary 3.7.4 *Let f be a nondecreasing, total recursive function such that*

$$s = \sum_{i \geq 1} 2^{-f(i)}$$

is a recursive real number and $s \leq 1$. Then there is an algorithm that computes $\chi(G)$ for recursive graphs by making only $f(n)$ serial queries to K.

Proof: This follows from Theorem 3.7.3 and Theorem 3.7.1. ▮

Theorem 3.7.5 *If there exists an oracle B and an algorithm that computes $\chi(G)$ for recursive graphs by making only $f(\chi(G))$ serial queries to B, then*

$$\sum_{i \geq 0} 2^{-f(i)} \leq 1.$$

The proof of this depends on ideas from Chapter 4. Therefore, we defer the proof to Appendix A.

Corollary 3.7.6 *Let*

$$f(n) = \sum_{1 \leq i \leq \log^* n} \log^{(i)}(n) - (\log \log e) \log^* n + O(1).$$

There exists no oracle B such that we can compute $\chi(G)$ for recursive graphs by making only $f(\chi(G))$ serial queries to B.

Proof: By contradiction. By the preceding theorem, $\sum_{i \geq 0} 2^{-f(i)} \leq 1$. However, in [Bei90], we showed that $\sum_{i \geq 0} 2^{-f(i)}$ diverges. ▮

3.8 Related Work

The $\mathrm{Q}_{||}(n, K)$ hierarchy has arisen previously in other contexts. Putnam [Put65] calls P a k-trial predicate if P is computed by a machine that changes its mind at most k times on any input.

Definition 3.8.1 [Putnam] P is a k-trial predicate if there exists a total recursive f such that

$$P(x_1, \ldots, x_n) \equiv (\lim_{y\to\infty} f(x_1, \ldots, x_n, y) = 1),$$

and there are at most k natural numbers y such that

$$f(x_1, \ldots, x_n, y) \neq f(x_1, \ldots, x_n, y+1).$$

Putnam did not examine the hierarchy of k-trial predicates; instead he considered the set of predicates that are k-trial predicates for some k. He proved "there exists a k such that P is a k-trial predicate if and only if P belongs to Σ_1^*, the smallest class containing the recursively enumerable predicates and closed under truth functions."

Ershov [Ers68a] defines the following classes:

Definition 3.8.2 [Ershov] Let $\mathcal{F}_0$ denote the set of one-one partial recursive one-place functions.

$$\Sigma_0^{-1} = \Pi_0^{-1} = \mathrm{SREC};$$

$$\Sigma_{n+1}^{-1} = \{X \mid (\exists f \in \mathcal{F}_0)(\exists Y \in \Pi_n^{-1})[X = f(Y)]\};$$

$$\Pi_{n+1}^{-1} = \text{co-}\Sigma_{n+1}^{-1} = \{X \mid (\exists Y \in \Sigma_{n+1}^{-1})[X = \mathsf{N} - Y]\}.$$

Ershov proves that $X \in \Sigma_n^{-1}$ if and only if X is of the form

$$R_1 - (R_2 - (R_3 \ldots - (R_{n-1} - R_n) \ldots))$$

where $R_1, \ldots, R_n$ are r.e. Thus $X \in \Sigma_n^{-1}$ if and only if X is m-reducible to PARITY_n^K. Ershov proves that $X \in \Sigma_{n+1}^{-1} \cap \Pi_{n+1}^{-1}$ if and only if $(x \in X)$ is an n-trial predicate. Ershov also proves that his classes form a hierarchy, *e.g.*, Σ_n^{-1} is properly contained in Σ_{n+1}^{-1}. In [Ers68b] and [Ers70], he finds two different techniques to extend his hierarchy over the ordinal numbers.

These ideas were treated more recently in [Eps79], which defines the n-r.e. sets. His definition can be considered a modification of Putnam's definition of the n-trial predicates in which $f(x_1, \ldots, x_n, 0)$ is required to be 0. Thus 1-r.e. sets are the same as r.e. sets. The class of n-r.e. sets is identical to Ershov's class, Σ_n^{-1}.

Epstein, Haas, and Kramer define the set of weakly n-r.e. sets, which is equal to $\Sigma_{n+1}^{-1} \cap \Pi_{n+1}^{-1}$. The weakly n-r.e. sets are equivalent to the n-trial predicates. In joint work with Gasarch and Hay [BGH89], we use this fact in order to show that $\mathrm{Q}_{||}(n, K)$ is equal to the set of weakly n-r.e. sets; we also use this fact in order to find different proofs of many of the results presented in Section 3.2 through Section 3.4.

Epstein, Haas, and Kramer also find a more intuitive method than Ershov's to extend the hierarchy over the ordinals [EHK81]. For $m \geq 1$, it turns out that $\mathrm{Q}_{||}([n]\omega^m, K)$ is identical to their class, $\nabla_{(n+1)\omega^m}$. If we define our hierarchy more subtly, and place different bounds on the number of parallel queries to be made depending on the results of the preceding queries, then we can refine our hierarchy so that it is identical with the one in [EHK81].

It is interesting that the $\mathrm{Q}_{||}(n, K)$ hierarchy arises in so many natural ways. For more related work see [Add65] and [Gol65].

Chapter 4

Nonrecursive Oracles

In Section 4.1, we give a new definition of computability that enables us to apply purely combinatorial techniques to our study of bounded-query classes. In Section 4.2, we prove the Nonspeedup Theorem, which says that 2^n queries to a nonrecursive oracle A cannot be answered by making only n queries to an oracle B. In Section 4.3, we show that $n+1$ queries to a nonrecursive oracle B allow us to compute more functions than n queries to B allow us to compute. In the remainder of the chapter, we investigate separation results for decision problems, and we define and study terse, superterse, and verbose sets.

4.1 Computability by a Set of Partial Recursive Functions

In this section, we define a new notion of computability that captures the information-theoretic aspects of n-query oracle computations. The new notion of computability is independent of the particular oracle being used, thus allowing us to apply purely combinatorial techniques to the study of bounded-query computations.

Definition 4.1.1 The partial function h is *computable by a set of n partial recursive functions* if there exist n partial recursive functions $g_1, \ldots, g_n$ such that

$$(\forall x)[\text{if } h(x) \text{ converges then } h(x) \in \{g_i(x) \mid 1 \le i \le n\}].$$

(If $g_i(x)$ does not converge then we exclude its value from the set above, by convention.)

Thus, the function h is computable by a set of n partial recursive functions if, for each x, we can effectively compute a list of length n that includes $h(x)$. Informally, we say that *there are only n possible values* for $h(x)$.

Thus, for example, every $0,1$-valued function is computable by a set of two partial recursive functions: let $g_i(x) = i - 1$. However, not every $0,1,2$-valued function is computable by a set of two partial recursive functions, because we can diagonalize in the standard way.

The next theorem implies that computability by a set of partial recursive functions captures the information-theoretic aspects of computability by an oracle.

Theorem 4.1.2

i. If there exists an oracle B such that $h \in \mathrm{FQ}(n, B)$, then h is computable by a set of 2^n partial recursive functions.

ii. If h is computable by a set of 2^n partial recursive functions, then there exists an oracle B such that $h \in \mathrm{FQ}_{||}(n, B)$. If h is total, then B is in $\mathrm{Q}(1, h)$

Proof:

i. Suppose that h is computed by an n-query B-machine M. There are only 2^n possible sequences of n oracle answers. For each i, let g_i simulate M by using the ith (lexically) sequence of oracle answers, instead of querying B. For each x, one of the sequences of oracle answers must be the correct one; therefore at least one of the g_i's correctly computes $h(x)$.

ii. Suppose that h is computable by a set of 2^n partial recursive functions, $g_1, \ldots, g_{2^n}$. Then n bits of information are sufficient to specify the first i such that $g_i(x) = h(x)$. We define B to be an oracle that provides those bits, *i.e.*, $(x, j) \in B$ if and only if the jth bit of the aforementioned $i - 1$ is equal to 1. Given an oracle for B, we compute $h(x)$ as follows: make n parallel queries to B in order to determine (one bit at a time) an i such that $h(x) = g_i(x)$; output $g_i(x)$. Thus $h \in \mathrm{FQ}_{||}(n, B)$. We determine membership in B as follows:

Step 1: Input (x, j).

Step 1.5: (* If h is total then we can skip this step. *) If $h(x)$ diverges then reject.

Step 2: Compute $h(x)$.

Step 3: Timeshare the computations of $g_1(x), \ldots, g_{2^n}(x)$ until one of them outputs the correct answer $h(x)$. Let $g_i(x)$ be the first to output the correct answer.

Step 4: If the jth bit of $i - 1$ is equal to 1 then accept; otherwise, reject.

If h is total, we omit step 1.5, so that $B \in \mathrm{Q}(1, h)$. If h is not total then step 1.5 is necessary so that B will be a set; in this case, B is r.e. in h. ∎

This theorem enables us to show that every total function h computable by making n serial queries to an oracle A can be computed by making n *parallel* queries to a different oracle B such that $B \in \mathrm{Q}(1, h)$.

Corollary 4.1.3 *If h is a total function in* $\mathrm{FQ}(n, A)$ *then there exists a set B in* $\mathrm{Q}(1, h)$ *such that h is in* $\mathrm{FQ}_{||}(n, B)$.

Proof: By Theorem 4.1.2(i), h is computable by a set of 2^n partial recursive functions. Therefore, by Theorem 4.1.2(ii), there is a set B in $\mathrm{Q}(1, h)$ such that h is in $\mathrm{FQ}_{||}(n, B)$. ∎

4.2 The Nonspeedup Theorem

Corollary 3.5.11 stated that $\mathrm{F}_2^K \notin \mathrm{FQ}(1, B)$ for any oracle B. We generalize that to a result proved independently by Gasarch [Gas86]: If A is any nonrecursive set, then two parallel queries to A cannot be answered by making only one query to any set B. In addition, we show that 2^n parallel queries to a nonrecursive set A cannot be answered by making only n serial queries to any set B. This is the strongest possible result, by Theorem 3.2.2.

Theorem 4.2.1 *If A is a nonrecursive set and B is any set, then*

$$\mathrm{F}_2^A \notin \mathrm{FQ}(1, B).$$

Proof: By contradiction. Assume that $\mathrm{F}_2^A \in \mathrm{FQ}(1, B)$. By Theorem 4.1.2(i), F_2^A is computable by a set of two partial recursive functions. That is, there exist partial recursive functions g_1, g_2 such that

$$(\forall x, y)[\mathrm{F}_2^A(x, y) \in \{g_1(x, y), g_2(x, y)\}].$$

Let π be the operator that projects an ordered pair onto its first component. We take two cases.

Case 1:

$$(\forall x)(\exists y)[\pi g_1(x, y) = \pi g_2(x, y)].$$

In this case the following is an algorithm to compute $\chi_A(x)$: Timeshare $g_1(x, y)$ and $g_2(x, y)$ for all y until we find a y_0 such that $\pi g_1(x, y_0) = \pi g_2(x, y_0)$. One of the two functions gives the right answer; since they agree, they both give the right answer. Therefore, $\chi_A(x) = \pi g_1(x, y_0)$.

Case 2:

$$(\exists x)(\forall y)[\pi g_1(x, y) \neq \pi g_2(x, y)].$$

In this case choose x_0 such that $\pi g_1(x_0, y) \neq \pi g_2(x_0, y)$ for all y. Let $c = \chi_A(x_0)$. Then the following is an algorithm to compute $\chi_A(y)$: Timeshare $g_1(x_0, y)$ and $g_2(x_0, y)$ until one of them produces an output whose first component is c. Since the other function disagrees with the one that produces c, the other function must be incorrect. Thus $\chi_A(y)$ is equal to the second component of the function that produces c for the first component.

In either case, χ_A is computable, so A is recursive. This contradiction proves the theorem. ∎

This theorem does not generalize in the obvious way. In fact, by Theorem 3.2.2, $\mathrm{F}^K_{2^n-1} \in \mathrm{FQ}(n, K)$. However, we can prove that this result for halting-problem oracles is tight, as part of a general result.

Lemma 4.2.2 *Let $m, n > 1$. If F^A_n is computable by a set of m partial recursive functions, then F^A_{n-1} is computable by a set of $m-1$ partial recursive functions.*

Proof: Assume that there exist $g_1, \ldots, g_m$ such that

$$(\forall x_1, \ldots, x_n)[\mathrm{F}^A_n(x_1, \ldots, x_n) \in \{g_i(x_1, \ldots, x_n) \mid 1 \le i \le m\}]. \qquad (5)$$

Let π be the operator that projects an n-tuple onto its first $n-1$ components. We take two cases.

Case 1:

$$(\forall x_1, \ldots, x_{n-1})(\exists x_n)(\exists (i_1 \neq i_2))[\pi g_{i_1}(x_1, \ldots, x_n) = \pi g_{i_2}(x_1, \ldots, x_n)]. \qquad (6)$$

In this case, for each input, two of the functions $g_1, \ldots, g_m$ agree on the first $n-1$ components. Thus we can define $m-1$ functions of $n-1$ variables that omit the repeated value. Formally, for $1 \le i \le m-1$, we define the function $g'_i(x_1, \ldots, x_{n-1})$, which is computed as follows:

Step 1: Input $\vec{x} = (x_1, \ldots, x_{n-1})$.

Step 2: Timeshare the computations of $g_j(\vec{x}, y)$ for all j and all y until we find j_1, j_2, and y such that $j_1 < j_2$ and

$$\pi g_{j_1}(\vec{x}, y) = \pi g_{j_2}(\vec{x}, y),$$

as guaranteed by equation (6).

Step 3: If $i < j_2$ then output $\pi g_i(\vec{x}, y)$; otherwise, output $\pi g_{i+1}(\vec{x}, y)$.

Thus g'_i is partial recursive for every i, and $g'_1, \ldots, g'_{m-1}$ take on the same set of values as $\pi g_1, \ldots, \pi g_m$, so

$$(\forall x_1, \ldots, x_{n-1})[\mathrm{F}^A_{n-1}(x_1, \ldots, x_{n-1}) \in \{g'_i(x_1, \ldots, x_{n-1}) \mid 1 \leq i \leq m-1\}].$$

Therefore F^A_{n-1} is computable by a set of $m-1$ partial recursive functions.

Case 2:

$$(\exists x_1, \ldots, x_{n-1})(\forall x_n)(\forall i_1 \neq i_2)[g_{i_1}(x_1, \ldots, x_n) \neq g_{i_2}(x_1, \ldots, x_n)]. \tag{7}$$

In this case, let $\vec{x} = (x_1, x_2, \ldots, x_{n-1})$ be the $(n-1)$-tuple whose existence is guaranteed by (7). Let $\vec{c} = \mathrm{F}^A_{n-1}(x_1, \ldots, x_{n-1})$. Then the following is an algorithm to compute $\chi_A(y)$: Timeshare $g_1(\vec{x}, y)$ through $g_m(\vec{x}, y)$, until one of them produces an output whose first $n-1$ components are $\vec{c}$. Then $\chi_A(y)$ is equal to the last component. Thus A is recursive. Therefore, F^A_n is computable by a set consisting of one partial recursive function.

▮

Lemma 4.2.3 (Nonspeedup) *If A is nonrecursive then F^A_n is not computable by a set of n partial recursive functions.*

Proof: By contradiction. Assume that F^A_n is computable by a set of n partial recursive functions. By repeated application of the previous lemma, we see that F^A_1 is computable by a set consisting of one partial recursive function. Therefore A is recursive. ▮

Theorem 4.2.4 (Nonspeedup) *If A is a nonrecursive set and B is any set then*

$$\mathrm{F}^A_{2^n} \notin \mathrm{FQ}(n, B).$$

Proof: By contradiction. Assume that $\mathrm{F}^A_{2^n} \in \mathrm{FQ}(n, B)$. Then $\mathrm{F}^A_{2^n}$ is computable by a set of 2^n partial recursive functions by Theorem 4.1.2(i). This contradicts the Nonspeedup Lemma. ▮

We relativize the definitions of the bounded query classes.

Definition 4.2.5

- $\mathrm{MQ}^C(n,A)$ is the set of machines with oracles for A and for C that make at most n queries to A and an unrestricted number of queries to C.

- $\mathrm{FQ}^C(n,A)$ is the set of partial functions that are computable by machines in $\mathrm{MQ}^C(n,A)$.

- $\mathrm{Q}^C(n,A)$ is the set of 0,1-valued total functions that are in $\mathrm{FQ}^C(n,A)$.

- $\mathrm{MQ}_{||}^C(n,A)$ is the set of machines with oracles for A and for C that make at most n queries to A, all queries being made in parallel, and an unrestricted number of serial queries to C.

- $\mathrm{FQ}_{||}^C(n,A)$ is the set of partial functions that are computable by machines in $\mathrm{MQ}_{||}^C(n,A)$.

- $\mathrm{Q}_{||}^C(n,A)$ is the set of 0,1-valued total functions that are in $\mathrm{FQ}_{||}^C(n,A)$.

Theorem 4.2.6 (Relativized Nonspeedup) *Let A be a set that is not recursive in the set C. Then for all B*

$$\mathrm{F}_{2^n}^A \notin \mathrm{FQ}^C(n,B).$$

Proof: The proof of the Nonspeedup Theorem relativizes. ▮

4.3 Separation Theorems

The Nonspeedup Theorem enables us to generalize Lemma 3.2.10 to arbitrary nonrecursive oracles. That is,

Lemma 4.3.1 *If A is nonrecursive then*

$$\mathrm{FQ}_{||}(n,A) \subset \mathrm{FQ}_{||}(n+1,A).$$

Proof: The containment follows from the definition of $\mathrm{FQ}_{||}$. Assume that the containment is not proper, so that $\mathrm{FQ}_{||}(n, A) = \mathrm{FQ}_{||}(n+1, A)$. Then $\mathrm{FQ}_{||}(m, A) = \mathrm{FQ}_{||}(n, A)$ for all $m \geq n$, by Observation 2.2.15(ii). In particular $\mathrm{FQ}_{||}(2^n, A) = \mathrm{FQ}_{||}(n, A)$. Therefore, $\mathrm{F}^A_{2^n} \in \mathrm{FQ}_{||}(2^n, A) = \mathrm{FQ}_{||}(n, A) \subseteq \mathrm{FQ}(n, A)$. By the Nonspeedup Theorem, A must be recursive. ∎

Theorem 4.3.2 (Parallel Separation) *If A is nonrecursive then there is a total function in* $\mathrm{FQ}_{||}(n+1, A) - \mathrm{FQ}_{||}(n, A)$.

Proof: F^A_{n+1} is a total function in $\mathrm{FQ}_{||}(n+1, A) - \mathrm{FQ}_{||}(n, A)$, by Lemma 4.3.1. ∎

Theorem 4.3.3 *If A is nonrecursive then*

$$\mathrm{FQ}(n, A) \subset \mathrm{FQ}(n+1, A)$$

Proof: The containment follows from the definition of FQ. Assume that the containment is not proper, so that $\mathrm{FQ}(n, A) = \mathrm{FQ}(n+1, A)$. Then $\mathrm{FQ}(m, A) = \mathrm{FQ}(n, A)$ for all $m \geq n$, by Observation 2.2.15. In particular $\mathrm{FQ}(2^n, A) = \mathrm{FQ}(n, A)$. Therefore $\mathrm{F}^A_{2^n} \in \mathrm{FQ}_{||}(2^n, A) \subseteq \mathrm{FQ}(2^n, A) = \mathrm{FQ}(n, A)$. By the Nonspeedup Theorem, A must be recursive. ∎

If A is not recursive, then $n+1$ serial queries to A allow us to compute more partial functions that n serial queries to A allow us to compute. The proof of the preceding theorem depends only on the Nonspeedup Theorem and Observation 2.2.11 (Composition). It is tempting to define bounded query classes of *total* functions and then try to generalize our proofs directly. Unfortunately, our proof of Observation 2.2.11 is not valid for total functions; although we do not know whether Observation 2.2.11 is true for total functions, we suspect that it is not. Since we cannot generalize our proofs directly to total functions, we use a more complicated technique to show that $n+1$ serial queries to A allow us to compute more total functions that n serial queries to A allow us to compute.

Assume that A is nonrecursive. If it were the case that $\mathrm{F}^A_{n+1} \notin \mathrm{FQ}(n, A)$ then F^A_{n+1} would be a total function in $\mathrm{FQ}(n+1, A) - \mathrm{FQ}(n, A)$. If it were the case that

$\mathrm{F}^A_{2^n-1} \in \mathrm{FQ}(n, A)$ then $\mathrm{F}^A_{2^n}$ would be a total function in $\mathrm{FQ}(n+1, A) - \mathrm{FQ}(n, A)$. Those two cases are the extremes, which suggest the general rule. By the Nonspeedup Theorem, for every nonrecursive set A and every natural number n, there is a largest number u such that $\mathrm{F}^A_u \in \mathrm{FQ}(n, A)$. Then F^A_{u+1} is a total function belonging to $\mathrm{FQ}(n+1, A) - \mathrm{FQ}(n, A)$. We formalize this proof below.

Theorem 4.3.4 (Serial Separation) *If A is nonrecursive, then there is a total function in* $\mathrm{FQ}(n+1, A) - \mathrm{FQ}(n, A)$.

Proof: Let A be a nonrecursive set, and let n be any natural number. Let

$$u = \max\{t \mid \mathrm{F}^A_t \in \mathrm{FQ}(n, A)\}.$$

By the Nonspeedup Theorem, the maximum exists (in fact it is less than 2^n), so u is well defined.

$$\begin{aligned} \mathrm{F}^A_{u+1} &\in \mathrm{FQ}(1, \mathrm{F}^A_u) \,\|\, \mathrm{FQ}(1, A) \\ &\subseteq \mathrm{FQ}(n, A) \,\|\, \mathrm{FQ}(1, A) \qquad \text{because } \mathrm{F}^A_u \in \mathrm{FQ}(n, A) \\ &\subseteq \mathrm{FQ}(n+1, A). \end{aligned}$$

Since u was chosen as the maximum t such that $\mathrm{F}^A_t \in \mathrm{FQ}(n, A)$, it follows that

$$\mathrm{F}^A_{u+1} \notin \mathrm{FQ}(n, A).$$

Therefore,

$$\mathrm{F}^A_{u+1} \in \mathrm{FQ}(n+1, A) - \mathrm{FQ}(n, A).$$

∎

4.4 Decision Problems

In Chapter 3 we established tradeoffs between serial and parallel queries to the halting problem, and we proved separation results for bounded query classes of decision problems solvable with an oracle for K. In this chapter we have proved several separation results for bounded query classes of functions computable with an arbitrary nonrecursive oracle. In this section, we consider bounded query classes of decision problems solvable with an arbitrary nonrecursive oracle, and we investigate possible generalizations of our previous results. We show that our previous results do not generalize, except in special cases.

By Corollary 3.3.2(i), n-tt reducibility to K is equivalent to n-parallel-query reducibility to K. Lachlan has shown that n-tt reducibility is different from n-parallel-query reducibility in the general case (see the discussion after Corollary 3.3.2). Thus, Corollary 3.3.2(i) does not generalize.

One might hope to generalize the separation theorems of Section 4.3 to apply to decision problems, instead of just to functions. Theorem 4.4.1 below states that the Parallel Separation Theorem (4.3.2) does not generalize to decision problems. In [Bei87d], we construct a nonrecursive set B such that $\mathrm{Q}(n, B) = \mathrm{Q}(1, B)$ for all n; therefore the Serial Separation Theorem (4.3.4) does not generalize to decision problems.

Theorem 4.4.1 *There exists a nonrecursive set B such that*

$$\mathrm{Q}_{||}(\omega, B) = \mathrm{Q}(1, B).$$

Proof: Let $B = \mathrm{PARITY}^K_\omega$.

$$\begin{aligned} \mathrm{Q}_{||}(\omega, B) &= \mathrm{Q}_{||}(\omega, \mathrm{PARITY}^K_\omega) \\ &\subseteq \mathrm{Q}_{||}(\omega, K) \qquad \text{because } \mathrm{PARITY}^K_\omega \in \mathrm{Q}_{||}(\omega, K) \\ &= \mathrm{Q}_{||}(1, \mathrm{PARITY}^K_\omega) \qquad \text{by Theorem 3.6.4} \\ &= \mathrm{Q}_{||}(1, B). \end{aligned}$$

Therefore $\mathrm{Q}_{||}(\omega, B) \subseteq \mathrm{Q}(1, B)$. The reverse containment is obvious. ∎

The following two theorems are from [Bei87d]:

Theorem 4.4.2 *If $B \in \mathrm{Q}(n, K)$ then*

$$\mathrm{Q}(n, B) \subset \mathrm{Q}(n+1, B) \text{ and } \mathrm{Q}_{||}(n, B) \subset \mathrm{Q}_{||}(n+1, B)$$

Theorem 4.4.3 *There exists a nonrecursive set B such that* $\mathrm{Q}(n, B) = \mathrm{Q}(1, B)$ *for every n.*

The following theorem is a generalization of Observation 2.2.15 to 0,1-valued partial functions.

Theorem 4.4.4 *Let B be a set and k a natural number.*

i. If there is no 0,1-valued partial function in $\mathrm{FQ}(k+1, B) - \mathrm{FQ}(k, B)$ *then there is no 0,1-valued partial function in* $\mathrm{FQ}(n, B) - \mathrm{FQ}(k, B)$ *for any n.*

ii. If there is no 0,1-valued partial function in $\mathrm{FQ}_{||}(2k, B) - \mathrm{FQ}_{||}(k, B)$ *then there is no 0,1-valued partial function in* $\mathrm{FQ}_{||}(n, B) - \mathrm{FQ}_{||}(k, B)$ *for any n.*

Proof:

i. By induction on n. By assumption, the claim is true for $n \leq k+1$. Suppose that the claim is true for some $n \geq k+1$. Let f be a 0,1-valued partial function that can be computed by an $(n+1)$-query B-machine M. Without loss of generality, assume that M makes exactly $n+1$ serial queries to B whenever M halts. For $i = 0, 1$, we define a function f_i computed as follows: simulate M until M is about to make its first serial query; complete the simulation assuming that the answer to M's first serial query is i. Then f_i is a 0,1-valued partial function in $\mathrm{FQ}(n, B)$; therefore, by the induction hypothesis, f_i is in $\mathrm{FQ}(k, B)$. We can compute f by making M's first query and then k more queries to simulate either M^0 or M^1, depending on the answer to the first query. Thus f is in $\mathrm{FQ}(k+1, B)$. By assumption f is in $\mathrm{FQ}(k, B)$.

ii. By induction on n. By assumption, the claim is true for $n \leq 2k$. Suppose that the claim is true for some $n \geq 2k$. Let f be a 0,1-valued partial function that can be computed by an $(n+1)$-parallel-query B-machine, M. Without loss of generality, assume that M makes exactly $n+1$ parallel queries whenever M halts. For $i = 0, 1$, we define a function f_i computed as follows: simulate M until M is about to make its round of $n+1$ parallel queries; make the last

n parallel queries; complete the simulation assuming that the answer to the first parallel query is i. Then f_i is a 0,1-valued partial function in $\mathrm{FQ}_{||}(n, B)$; therefore, by the induction hypothesis, f_i is in $\mathrm{FQ}_{||}(k, B)$. Let

$$g_i(x) = \begin{cases} f_i(x) & \text{if the answer to } M\text{'s first parallel query is } i \\ 0 & \text{if the answer to } M\text{'s first parallel query is } 1 - i \\ \text{undefined} & \text{if } M \text{ makes no queries.} \end{cases}$$

Then $f(x) = g_0(x) \vee g_1(x)$, and each g_i is a partial 0,1-valued function in $\mathrm{FQ}_{||}(k+1, B)$. By assumption g_i is in $\mathrm{FQ}_{||}(k, B)$. Therefore f is in $\mathrm{FQ}_{||}(2k, B)$, which is a subset of $\mathrm{FQ}_{||}(k, B)$ by assumption.

∎

We prove some separation results in special cases.

Lemma 4.4.5 *Let B be a set such that $K \in \mathrm{Q}(j, B)$. Then, for all k,*

$$\mathrm{Q}(k, B) \subset \mathrm{Q}(jk + j + k, B).$$

Proof: The containment is obvious. The following algorithm solves the halting problem for k-query B-machines:

Step 1: Input a k-query B-machine M. If M is not in standard form then reject. Normalize M so that M makes exactly k serial queries to B whenever M halts.

Step 2: For $i = 1, \ldots, k$ do the following:

(a) Query K to determine if M is going to make another query to B.

(b) If so, then query B to determine the answer; otherwise, reject.

Step 3: Query K to determine if the remainder of M's computation terminates.

This algorithm makes $k + 1$ serial queries to K and k serial queries to B. Since $K \in \mathrm{FQ}(j, B)$, the halting problem for k-query B-machines is in $\mathrm{Q}(jk + j + k, B)$. By Lemma 3.2.5, the halting problem for k-query B-machines is not in $\mathrm{Q}(k, B)$. ∎

Theorem 4.4.6 *Let B be a set such that $K \in \mathrm{Q}(j,B)$. Then, for all k, there is a 0,1-valued partial function in* $\mathrm{FQ}(k+1,B) - \mathrm{FQ}(k,B)$.

Proof: Proof by contradiction. Assume that there is no 0,1-valued partial function in $\mathrm{FQ}(k+1,B) - \mathrm{FQ}(k,B)$. By Theorem 4.4.4, there is no 0,1-valued partial function in $\mathrm{FQ}(n,B) - \mathrm{FQ}(k,B)$ for any n. In particular there is no 0,1-valued partial function in $\mathrm{FQ}(jk+j+k,B) - \mathrm{FQ}(k,B)$, so $\mathrm{Q}(jk+j+k,B) = \mathrm{Q}(k,B)$. This contradicts Lemma 4.4.5. ∎

Next, we show that Lemma 3.2.1 does not generalize to arbitrary nonrecursive oracles, even if we consider only decision problems.

Theorem 4.4.7 *There exists a nonrecursive set B such that*

$$\mathrm{Q}_{||}(\omega, B) \subset \mathrm{Q}(2, B).$$

Proof: Let $B = \mathrm{PARITY}_{\omega}^{K}$.

$$\begin{aligned}
\mathrm{Q}_{||}(\omega, \mathrm{PARITY}_{\omega}^{K}) &= \mathrm{Q}_{||}(1, \mathrm{PARITY}_{\omega}^{K}) \quad \text{as shown in the proof of Theorem 4.4.1} \\
&= \mathrm{Q}_{||}(\omega, K) \quad \text{by Theorem 3.6.4} \\
&\subset \mathrm{Q}_{||}(\omega, K) \circ \mathrm{FQ}(1, K) \quad \text{by Theorem 3.6.9} \\
&= \mathrm{Q}_{||}(1, \mathrm{PARITY}_{\omega}^{K}) \circ \mathrm{FQ}(1, K) \quad \text{by Theorem 3.6.4} \\
&\subseteq \mathrm{Q}_{||}(1, \mathrm{PARITY}_{\omega}^{K}) \circ \mathrm{FQ}(1, \mathrm{PARITY}_{\omega}^{K}) \\
&\ \text{because } K \leq_{\mathrm{m}} \mathrm{PARITY}_{\omega}^{K} \\
&= \mathrm{Q}(2, \mathrm{PARITY}_{\omega}^{K}).
\end{aligned}$$

∎

The following theorem is from [BGGO93];

Theorem 4.4.8 *There exists an oracle B such that*

$$\mathrm{Q}_{||}(n+1, B) \not\subseteq \mathrm{Q}(n, B).$$

Thus Lemma 3.1.6 does not generalize to arbitrary nonrecursive oracles, even if we consider only decision problems.

4.5 Terse and Superterse Sets

A set B is terse, as defined in [BGGO93], if it is not possible to answer n parallel queries to B by making only $n-1$ serial queries to B.

Definition 4.5.1 A set B is *terse* if

$$(\forall n)[\mathrm{F}_n^B \notin \mathrm{FQ}(n-1, B)].$$

In [BGGO93], it was shown that every r.e. Turing degree[1] contains an r.e. terse set. A set B is superterse if it is not possible to answer n parallel queries to B by making only $n-1$ serial queries to *any* oracle.

Definition 4.5.2 A set B is *superterse* if

$$(\forall A)(\forall n)[\mathrm{F}_n^B \notin \mathrm{FQ}(n-1, A)].$$

Theorem 4.5.3 *If B is r.e., then B is not superterse.*

Proof:

$$\begin{aligned} \mathrm{F}_{2^k-1}^B &\in \mathrm{FQ}_{||}(2^k-1, K) \qquad \text{because } B \leq_m K \\ &\subseteq \mathrm{FQ}(k, K). \end{aligned}$$

▮

The following corollary appears in [BGGO93]:

Corollary 4.5.4 *There exist oracles that are terse but not superterse.*

Proof: As shown in [BGGO93], there exist r.e. terse sets. ▮

[1] If r is a reducibility such that r-reducibility is reflexive and transitive, then we define the corresponding equivalence relation r-equivalence by $A \equiv_r B$ if and only if $A \leq_r B$ and $B \leq_r A$. An r-degree is an equivalence class of r-equivalence. See [Rog67, Soa87]. A degree is r.e. if it contains an r.e. set.

The Nonspeedup Theorem provides us with a tool for proving that an oracle is superterse.

Theorem 4.5.5 *If A is nonrecursive and*

$$(\forall n)[\mathrm{F}^A_{2^n-1} \in \mathrm{FQ}_{||}(n, B)]$$

then B is superterse.

Proof: Suppose that B is not superterse, so that, for some set C and some positive integer n

$$\mathrm{F}^B_n \in \mathrm{FQ}(n-1, C). \tag{8}$$

By assumption

$$\begin{aligned} \mathrm{F}^B_{2^n-1} &\in \mathrm{FQ}_{||}(n, B) \\ &\subseteq \mathrm{FQ}(n-1, C) \qquad \text{by equation (8),} \end{aligned}$$

contradicting the Nonspeedup Theorem. ∎

Lemma 4.5.6

$$\mathrm{F}^K_{2^n-1} \in \mathrm{FQ}_{||}(n, \mathrm{PARITY}^K_\omega).$$

Proof: Because $\mathrm{F}^K_{2^n-1} \in \mathrm{Q}(1, \#^K_{2^n-1})$ by Lemma 3.1.5, it suffices to show that

$$\#^K_{2^n-1} \in \mathrm{FQ}_{||}(n, \mathrm{PARITY}^K_\omega).$$

Let $t = \#^K_{2^n-1}(x_1, \ldots, x_{2^n-1})$. Then t is an n-bit nonnegative integer. Since PARITY^K_ω is 1-query complete for $\mathrm{Q}(\omega, K)$, each bit of t can be determined by making a single query to PARITY^K_ω. ∎

Corollary 4.5.7 PARITY^K_ω *is superterse.*

Proof: This follows from Lemma 4.5.6 and Theorem 4.5.5. ∎

Thus, there exists a superterse set.

4.6 Verbose Sets

The Nonspeedup Theorem provides a quantitative bound on how "nonterse" a nonrecursive set can be. We say that a set is verbose if it is as nonterse as the Nonspeedup Theorem allows, *i.e.*, a set B is verbose if $2^n - 1$ parallel queries to B can be answered by making only n serial queries to B, for all n.

Definition 4.6.1 A set B is *verbose* if

$$(\forall n)[\mathrm{F}^B_{2^n-1} \in \mathrm{FQ}(n, B)].$$

For example, K is verbose, by Theorem 3.2.2. We define semiverboseness analogously to superterseness.

Definition 4.6.2 A set B is *semiverbose* if

$$(\exists A)(\forall n)[\mathrm{F}^B_{2^n-1} \in \mathrm{FQ}(n, A)].$$

For example, all r.e. sets are semiverbose, as shown in the proof of Theorem 4.5.3. Since some r.e. sets are also terse, there exist sets that are semiverbose but not verbose.

In [BGGO93], it was shown that every truth-table degree contains a verbose set. The following proof is based on the proof in that paper and on Jockusch's construction of a semirecursive[2] set [Joc68].

Theorem 4.6.3 *If A is any nonrecursive set, then there exists a verbose set B that is truth-table equivalent to A.*

Proof: Let f be a recursive function that maps the natural numbers 1-1 onto the set of finite strings over the alphabet $\{0, 1\}$; then f^{-1} is a recursive function. We write $s_1 < s_2$ to denote that the (possibly infinite) string s_1 precedes the (possibly infinite) string s_2 in lexicographic order. Let α be the infinite string, $a_1 a_2 \cdots$, where $a_i = \chi_A(i)$. Let $B = \{x \mid f(x) < \alpha\}$.

[2] A set B is semirecursive if there exists a total recursive 0,1-valued function f such that if $f(x, y) = 1$ then $\chi_B(x) \geq \chi_B(y)$ and if $f(x, y) = 0$ then $\chi_B(x) \leq \chi_B(y)$.

First we show that $B \leq_{\mathrm{tt}} A$. We write $\alpha[n]$ to denote the finite string consisting of the first n characters of α. The following algorithm computes $\chi_B(x)$: let n be the length of the string $f(x)$; make n parallel queries to A in order to determine $\alpha[n]$; if $f(x) \leq \alpha[n]$ then output 1, else output 0.

Second we show that $A \leq_{\mathrm{tt}} B$. We determine whether n is in B as follows: For each string s of length n, query B to determine if $f^{-1}(s) \in B$, thereby determining whether $s < \alpha$; this determines the first n bits of α, and in particular it determines the nth bit of α, which is $\chi_B(n)$.

Finally we show that $2^n - 1$ parallel queries to B can be answered by making only n serial queries to B. We define a linear ordering on the natural numbers as follows: for $x, y \in \mathsf{N}$, let $x \prec y$ if and only if $f(x) < f(y)$. We compute $\mathrm{F}^B_{2^n-1}$ as follows: given $x_1, x_2, \ldots, x_{2^n-1}$, order them according to $\prec$; assume (by renumbering) that $x_1 \prec x_2 \prec \ldots \prec x_{2^n-1}$; use binary search ($n$ queries) to find the largest i such that $x_i \in B$; then we know that $x_1, \ldots, x_i$ are in B and that $x_{i+1}, \ldots, x_n$ are not in B. ▮

We included the proof of Theorem 4.6.3 because it leads to the following corollary:

Theorem 4.6.4 *There is a superterse set in every nonrecursive truth-table degree.*

Proof: We use the notation from the proof of Theorem 4.6.3. Let $S = \mathrm{PARITY}^B_\omega$, and let $t = \#^B_{2^n-1}(x_1, \ldots, x_{2^n-1})$.

Then n parallel queries to S allow us to determine t, as follows: Assume that $x_1 \prec x_2 \prec \ldots \prec x_{2^n-1}$. For $0 \leq i < n$, we can determine the ith bit of t by asking if an odd number of $\{x_{j2^i} \mid 1 \leq j < 2^{n-i}\}$ are in B. Once we know t, we know that $x_1, \ldots, x_t$ are in B and that $x_{t+1}, \ldots, x_{2^n-1}$ are not in B. Thus

$$\mathrm{F}^B_{2^n-1} \in \mathrm{FQ}_{||}(n, S).$$

By Theorem 4.5.5, S is superterse. ▮

4.7 $\#^A_{2^n} \in ?\ \mathrm{FQ}(n, B)$

If B is defined as in the proof of Theorem 4.6.3, then $\mathrm{F}^B_n \in \mathrm{FQ}(1, \#^B_n)$, for all n. Therefore, by the Nonspeedup Theorem, for every set S and natural number n,

$$\#^B_{2^n} \notin \mathrm{FQ}(n, S).$$

This leads us to conjecture the following, stronger version of the Nonspeedup Theorem.

Conjecture 4.7.1 *Let A be any nonrecursive set. For every set B and natural number n*

$$\#^A_{2^n} \notin \mathrm{FQ}(n, B).$$

Theorem 4.7.2 *Let A be any set. Assume that there exist sets B and C such that $B \leq_{\mathrm{m}} A$, B is r.e. in C, and B is not recursive in C. Then for every set S and natural number n*

$$\#^A_{2^n} \notin \mathrm{FQ}(n, S).$$

Proof: By assumption, $B \leq_{\mathrm{m}} A$. Therefore $\#^B_k \in \mathrm{FQ}(1, \#^A_k)$, for all k. In particular,

$$\#^B_{2^n} \in \mathrm{FQ}(1, \#^A_{2^n}).$$

By assumption, B is r.e. in C. Therefore, because Lemma 3.1.4 relativizes,[3] $\mathrm{F}^B_k \in \mathrm{FQ}^C(1, \#^B_k)$, for all k. In particular,

$$\mathrm{F}^B_{2^n} \in \mathrm{FQ}^C(1, \#^B_{2^n}).$$

In order to obtain a contradiction, suppose that

$$\#^A_{2^n} \in \mathrm{FQ}(n, S).$$

Then, by transitivity,

$$\mathrm{F}^B_{2^n} \in \mathrm{FQ}^C(n, S).$$

Since B is not recursive in C, this contradicts the relativized Nonspeedup Theorem. ∎

[3] See [Rog67, Soa87] for a discussion of relativizations.

Corollary 4.7.3 *Let A be nonrecursive. Assume that there exists C such that A is r.e. in C, but not recursive in C. Then for every set S and natural number n*

$$\#_{2^n}^A \notin \mathrm{FQ}(n, S).$$

Proof: Since $A \leq_{\mathrm{m}} A$, we can let $B = A$ in the preceding theorem. ∎

The hypothesis of Corollary 4.7.3 is not true for every nonrecursive set A. (For example, let A equal the recursive join [Rog67, Soa87] of K and $\bar{K}$. If A is r.e. in C, then $\bar{A}$ is r.e. in C because $\bar{A} \leq_m A$. Therefore A is recursive in C.) Thus Corollary 4.7.3 is not powerful enough to establish Conjecture 4.7.1. In fact, Owings has pointed out[4] that even Theorem 4.7.2 is not powerful enough to establish Conjecture 4.7.1.

Corollary 4.7.4 *Let A be nonrecursive. Assume that there exists a nonrecursive r.e. set B such that $B \leq_{\mathrm{m}} A$. Then for every set S and natural number n*

$$\#_{2^n}^A \notin \mathrm{FQ}(n, S).$$

Proof: Let $C = \emptyset$ in Theorem 4.7.2. ∎

Corollary 4.7.5 *If A is a nonrecursive set in $\mathrm{Q}(k, K)$ then for every set S and natural number n*

$$\#_{2^n}^A \notin \mathrm{FQ}(n, S).$$

Proof: In [BGH89], we have shown that if $A \in \mathrm{Q}(k, K)$ then A is $(2^k - 1)$-r.e. If A is a nonrecursive n-r.e. set then Epstein, Haas, and Kramer have shown [EHK81, Theorem 3][5] that there exists a nonrecursive r.e. set B such that $B \leq_m A$ or $B \leq_{\mathrm{m}} \bar{A}$.

If $B \leq_m A$ then $\#_{2^n}^A \notin \mathrm{FQ}(n, S)$ by Corollary 4.7.4. Otherwise $B \leq_{\mathrm{m}} \bar{A}$, so $\#_{2^n}^{\bar{A}} \notin \mathrm{FQ}(n, S)$. Since $\#_{2^n}^{\bar{A}} \in \mathrm{FQ}(1, \#_{2^n}^A)$ by Observation 2.2.9, it follows that $\#_{2^n}^A \notin \mathrm{FQ}(n, S)$. ∎

[4] Choose a set A such that the Turing degree of A is minimal. If $B \leq_{\mathrm{m}} A$ then the degree of B must also be minimal. In [Soa87, p. 124] it is shown that no nonzero r.e. degree is minimal, as a corollary to the Sacks Splitting Theorem. Relativizing that proof, we obtain that if B is r.e. in C but not recursive in C then the degree of B is not minimal. Thus the hypothesis of Theorem 4.7.2 is not true for elements of minimal Turing degrees.

[5] They only claim to prove that $B \leq_{\mathrm{T}} A$; however their proof sketch yields the stronger result. A more detailed proof sketch appears in [Eps79, Theorem 7 on page 152]

Lemma 4.7.6

$$\mathrm{F}_n^A \in \mathrm{FQ}(1, \#_{2^n-1}^A)$$

Proof: In order to determine which of x_1 through x_n are in A, make 2^{i-1} copies of x_i for each i. Ask how many of those $2^n - 1$ strings are in A. Then $x_i \in A$ if and only if the ith bit of that answer is one. ∎

The next theorem is our most concrete progress on Conjecture 4.7.1:

Theorem 4.7.7 *Let A be any nonrecursive set. For every set B and natural number n*

$$\#_{2^{2^n}-1}^A \notin \mathrm{FQ}(n, B).$$

Proof: By contradiction. Suppose that

$$\#_{2^{2^n}-1}^A \in \mathrm{FQ}(n, B).$$

By Lemma 4.7.6,

$$\begin{aligned} \mathrm{F}_{2^n}^A &\in \mathrm{FQ}(1, \#_{2^{2^n}-1}^A) \\ &\subseteq \mathrm{FQ}(n, B) \qquad \text{by assumption,} \end{aligned}$$

which contradicts the Nonspeedup Theorem. ∎

The results in this section have been substantially improved by Owings, who proves two difficult theorems in [Owi89]. He has shown that if $\#_2^B$ is computable by a set of two partial recursive functions, then B is recursive. In addition, he has shown that if $\#_n^B$ is computable by a set of n partial recursive functions then B is recursive in K. By Corollary 4.7.5, B cannot be a nonrecursive set in $\mathrm{Q}(k, K)$ for any k. Thus Conjecture 4.7.1 is open only for sets that are Turing reducible to K, but not bounded-query[6] reducible to K.

[6] In [Sac61], Sacks has constructed a minimal Turing degree below $\mathbf{0}'$. Therefore Theorem 4.7.2 will not suffice (see footnote 4) to establish the remaining cases of Conjecture 4.7.1.

4.8 Quantifying Verboseness

So far, the only nonterse sets we have seen are verbose sets, and the only nonsuperterse sets we have seen are semiverbose sets. In this section, we exhibit some sets that are neither terse nor semiverbose. We also prove that if a set fails to be terse, then it fails in a strong way to be terse; if a set fails to be superterse then it comes within a constant factor of being semiverbose.

Definition 4.8.1

- A set B is *k-verbose* if

$$\mathrm{F}_n^B \in \mathrm{FQ}(k \log n + O(1), B).$$

- A set B is *k-semiverbose* if there exists a set A such that

$$\mathrm{F}_n^B \in \mathrm{FQ}_{||}(k \log n + O(1), A).$$

(The constant term $O(1)$ is allowed to depend in k.)

The next theorem is a useful tool for proving that a set is not semiverbose.

Theorem 4.8.2 *If there exists a nonrecursive set A such that*

$$\mathrm{F}_{2^n}^A \in \mathrm{FQ}_{||}(m, B)$$

then for every set C

$$\mathrm{F}_m^B \notin \mathrm{FQ}(n, C).$$

Proof: By contradiction. Assume that $\mathrm{F}_{2^n}^A \in \mathrm{FQ}_{||}(m, B)$ and $\mathrm{F}_m^B \in \mathrm{FQ}(n, C)$. Then $\mathrm{F}_{2^n}^A \in \mathrm{FQ}(n, C)$, which contradicts the Nonspeedup Theorem. ∎

Theorem 4.8.3 *For every $k \geq 2$, PARITY_k^K is 1-semiverbose, but not semiverbose.*

Proof:

$$
\begin{aligned}
\mathrm{F}_n^{\mathrm{PARITY}_k^K} &\in \mathrm{FQ}_{||}(n, \mathrm{PARITY}_k^K) \\
&\subseteq \mathrm{FQ}_{||}(kn, K) \qquad \text{because } \mathrm{PARITY}_k^K \in \mathrm{Q}_{||}(k, K) \\
&\subseteq \mathrm{FQ}(\lceil \log(kn+1) \rceil, K) \qquad \text{by Lemma 3.1.6} \\
&\subseteq \mathrm{FQ}(\lceil \log(kn+1) \rceil, \mathrm{PARITY}_k^K) \qquad \text{because } K \leq_{\mathrm{m}} \mathrm{PARITY}_k^K.
\end{aligned}
$$

Since $\log(kn+1) = \log n + O(1)$, we conclude that PARITY_k^K is 1-semiverbose. Let $m = k(n-k+2)-1$. The following algorithm computes $\#_m^K$:

Step 1: Input $\vec{x} = (x_1, \ldots, x_m)$;

Step 2: Construct y_i such that $y_i \in K$ if and only if $(i; \vec{x}) \in \mathrm{GEQ}^K$. (* Then $\#_m^K(\vec{y}) = \#_m^K(\vec{x})$. *)

Step 3: Perform the following two computations in parallel:

(a) Let $s = \#_{k-1}^K(y_{n-k+2}, y_{2(n-k+2)}, \ldots, y_{t(n-k+2)}, \ldots, y_{(k-1)(n-k+2)})$.
(* At this point we know that $s(n-k+2) \leq \#_m^K(\vec{x}) < (s+1)(n-k+2)$. *)

(b) For $1 \leq i < n-k+2$ do the following:

let $\vec{v}_i = (y_i, y_{i+n-k+2}, \ldots, y_{i+t(n-k+2)}, \ldots, y_{i+(k-1)(n-k+2)})$,
and let $s' = \#_{n-k+1}^{\mathrm{PARITY}_k^K}(\vec{v}_1, \ldots, \vec{v}_{n-k+1})$.

(* If s is even then $s' = \#_{n-k+1}^K(\vec{v}_s)$. If s is odd then $s' = \#_{n-k+1}^{\bar{K}}(\vec{v}_s)$. *)

Step 4: If s is even then output $s(n-k+2)+s'$.
Otherwise, output $s(n-k+2)+n-k+1-s'$.

We make $k-1$ parallel queries to K in step 3(a), and simultaneously we make $n-k+1$ parallel queries to PARITY_k^K in step 3(b). Thus

$$
\begin{aligned}
\#_m^K &\in \mathrm{FREC} \circ (\mathrm{FQ}_{||}(k-1, K) \parallel \mathrm{FQ}_{||}(n-k+1, \mathrm{PARITY}_k^K)) \\
&\subseteq \mathrm{FREC} \circ (\mathrm{FQ}_{||}(k-1, \mathrm{PARITY}_k^K) \parallel \mathrm{FQ}_{||}(n-k+1, \mathrm{PARITY}_k^K)) \\
&\qquad \text{because } K \leq_{\mathrm{m}} \mathrm{PARITY}_k^K \\
&= \mathrm{FQ}_{||}(n, \mathrm{PARITY}_k^K) \qquad \text{by Corollary 2.2.14.}
\end{aligned}
$$

Therefore by Lemma 3.1.5(ii),

$$\mathrm{F}_m^K \in \mathrm{FQ}_{||}(n, \mathrm{PARITY}_k^K).$$

Equivalently,

$$\mathrm{F}_{k(n-k+2)-1}^K \in \mathrm{FQ}_{||}(n, \mathrm{PARITY}_k^K).$$

Therefore, by Theorem 4.8.2, for every set C

$$\mathrm{F}_n^{\mathrm{PARITY}_k^K} \notin \mathrm{FQ}(\lfloor \log{(k(n-k+2)-1)} \rfloor, C).$$

For large n

$$\lceil \log{(n+1)} \rceil < \lfloor \log{(k(n-k+2)-1)} \rfloor,$$

so PARITY_k^K is not semiverbose. ∎

Tighter upper and lower bounds are possible.

Corollary 4.8.4

- *There is a set that is neither terse nor semiverbose.*
- *There is a set that is neither terse nor verbose.*

Proof: The preceding theorem showed that PARITY_2^K is such a set. ∎

In studying oracles that are neither terse nor verbose, one might look for an oracle B such that for every sufficiently large k,

$$\mathrm{F}_k^B \in \mathrm{FQ}(k-1, B) - \mathrm{FQ}(k-2, B).$$

However, if

$$\mathrm{F}_k^B \in \mathrm{FQ}(k-1, B)$$

then for every n,

$$\begin{aligned}
\mathrm{F}_{nk}^B &\in \underbrace{\mathrm{FQ}((k-1), B) \,\|\, \cdots \,\|\, \mathrm{FQ}(k-1, B)}_{n} \\
&\subseteq \underbrace{\mathrm{FQ}((k-1), B) \circ \cdots \circ \mathrm{FQ}(k-1, B)}_{n} \\
&\ \text{by Observation 2.2.17} \\
&= \mathrm{FQ}(n(k-1), B).
\end{aligned}$$

Thus, if B is not terse, then there exists $r < 1$ such that

$$\mathrm{F}_n^B \in \mathrm{FQ}(rn, B),$$

for sufficiently large n. In fact, we can prove an even stronger result.

Theorem 4.8.5 *If* $\mathrm{F}_j^A \in \mathrm{FQ}(k, A)$ *then for every* $t \geq 0$

$$\mathrm{F}_{j^t}^A \in \mathrm{FQ}(k^t, A).$$

Proof: By induction on t. The theorem is identically true in the base case ($t = 0$). Assume that the theorem is true for $t - 1$.

$$\begin{aligned}
F_{j^t}^A &\in \underbrace{\mathrm{FQ}_{\|}(j^{t-1}, A) \,\|\, \cdots \,\|\, \mathrm{FQ}_{\|}(j^{t-1}, A)}_{j} \\
&\quad \text{because we can ask the queries in } j \text{ groups of } j^{t-1} \text{ queries} \\
&\subseteq \underbrace{\mathrm{FQ}(k^{t-1}, A) \,\|\, \cdots \,\|\, \mathrm{FQ}(k^{t-1}, A)}_{j} \qquad \text{by the induction hypothesis} \\
&\subseteq \underbrace{\mathrm{FQ}_{\|}(j, A) \circ \cdots \circ \mathrm{FQ}_{\|}(j, A)}_{k^{t-1}} \qquad \text{by Observation 2.2.17} \\
&\subseteq \underbrace{\mathrm{FQ}(k, A) \circ \cdots \circ \mathrm{FQ}(k, A)}_{k^{t-1}} \qquad \text{by assumption} \\
&= \mathrm{FQ}(k(k^{t-1}), A) \qquad \text{by Observation 2.2.11} \\
&= \mathrm{FQ}(k^t, A).
\end{aligned}$$

∎

Corollary 4.8.6 *If A is not terse, then there exists $r < 1$ such that*

$$\mathrm{F}_n^A \in \mathrm{FQ}(n^r, A),$$

for sufficiently large n.

Proof: Suppose that $\mathrm{F}_{k+1}^A \in \mathrm{FQ}(k, A)$. Choose t such that

$$(k+1)^t < n \leq (k+1)^{t+1}.$$

Then

$$\begin{aligned} \mathrm{F}_n^A &\in \mathrm{FQ}_{\|}((k+1)^{t+1}, A) \\ &\subseteq \mathrm{FQ}(k^{t+1}, A) \qquad \text{by the preceding Theorem.} \end{aligned}$$

Furthermore, $t < \log_{k+1} n$, because we chose t such that $(k+1)^t < n$. Therefore,

$$\begin{aligned} k^{t+1} &< k^{1+\log_{k+1} n} \\ &= k(k^{\log_{k+1} n}) \\ &= k(n^{\log_{k+1} k}) \\ &< n^{\frac{1}{2}(1+\log_{k+1} k)}, \end{aligned}$$

for sufficiently large n. Consequently,

$$\mathrm{F}_n^A \in \mathrm{FQ}(n^{\frac{1}{2}(1+\log_{k+1} k)}, A),$$

for sufficiently large n. ▮

Later in this section, we prove an even stronger result about semiverboseness. If k queries to A can be answered by making fewer than k queries to a second oracle, then by Theorem 4.1.2(i) there are fewer than 2^k possible answers to any k queries to A. Therefore, given n queries, there are fewer than 2^k possible answers to any choice of k of those n queries. In this section, we prove a combinatorial lemma[7] that shows that, in that case, there are at most $S(n,k)$ possible answers to the entire list of n queries, where

$$S(n,k) = \sum_{i=0}^{k-1} \binom{n}{i}.$$

We write Σ_k^n to denote the set of k-element subsequences of $1, \ldots, n$.

Definition 4.8.7

$$\Sigma_k^n = \{(\sigma_1, \ldots, \sigma_k) \mid 1 \le \sigma_1 < \cdots < \sigma_k \le n\}.$$

[7]The author has recently been referred to [COS75] by Clarke, Owings, and Spriggs. Our combinatorial lemma is essentially a restatement of their theorem on simultaneous m-splitting trees. Our proof is significantly simpler.

Definition 4.8.8 Let $\vec{\sigma}$ be an element of Σ_k^n.

i. If $\vec{x} = (x_1, \ldots, x_n)$ is an n-tuple then

$$\vec{x}\,(\vec{\sigma}\,) = (x_{\sigma_1}, \ldots, x_{\sigma_k}).$$

ii. If X is a set of n-tuples then

$$X(\vec{\sigma}\,) = \{\vec{x}\,(\vec{\sigma}\,) \mid \vec{x} \in X\}.$$

For example $(17, 60, 27, 7, 6, 60)(1, 2, 3) = (17, 60, 27)$. Intuitively, $X(\vec{\sigma}\,)$ is the projection of the set X onto the k components indicated by the sequence $\vec{\sigma}$. For example, if we write $\{0, 1, 2\}^n$ to denote the set of all n-tuples whose components are in $\{0, 1, 2\}$, then $\{0, 1, 2\}^j(1, \ldots, k) = \{0, 1, 2\}^k$, provided that $k \leq j$.

Lemma 4.8.9 *Let X be a set of n-tuples of bits, and let k be an integer such that $1 \leq k \leq n$. If, for all $\vec{\sigma}$ in Σ_k^n*

$$\mathrm{card}(X(\vec{\sigma}\,)) < 2^k)$$

then

$$\mathrm{card}(X) \leq S(n, k).$$

Proof: By induction on n. Assume that the lemma is true for $1, \ldots, n-1$. Let X be a set of n-tuples of bits. The lemma is true when $k = n$ because $S(k, k) = 2^k - 1$. The lemma is true when $k = 1$, because all elements of X agree on each component so that X has only one element. Let $1 < k < n$.

Let $\vec{\sigma}$ be an element of Σ_k^{n-1}. Then $\vec{\sigma}$ is also an element of Σ_k^n, so by assumption,

$$\mathrm{card}(X(\vec{\sigma}\,)) < 2^k.$$

Because $\vec{\sigma}$ is a subsequence of $1, \ldots, n-1$,

$$X(\vec{\sigma}\,) = X(1, \ldots, n-1)(\vec{\sigma}\,).$$

Therefore,

$$\mathrm{card}(X(1, \ldots, n-1)(\vec{\sigma}\,)) < 2^k.$$

By the induction hypothesis,

$$\mathrm{card}(X(1,\ldots,n-1)) \le S(n-1,k).$$

If $(\vec{y}\,,0)$ or $(\vec{y}\,,1)$ is an element of X then, $\vec{y}$ must be an element of $X(1,\ldots,n-1)$. Let Y be the set of $(n-1)$-tuples $\vec{y}$ such that both $(\vec{y}\,,0)$ and $(\vec{y}\,,1)$ are elements of X. Then

$$\mathrm{card}(X) = \mathrm{card}(X(1,\ldots,n-1)) + \mathrm{card}(Y).$$

Let $\vec{\sigma}$ be an element of Σ^{n-1}_{k-1}. Then we claim that

$$\mathrm{card}(Y(\vec{\sigma}\,)) < 2^{k-1}.$$

Suppose not. Then

$$\mathrm{card}(Y(\vec{\sigma}\,)) = 2^{k-1}.$$

If $\vec{y} \in Y$ then $(\vec{y}\,,0) \in X$ and $(\vec{y}\,,1) \in X$. Therefore, if $z \in Y(\vec{\sigma}\,)$ then $(\vec{z}\,,0) \in X((\vec{\sigma}\,,n))$ and $(\vec{z}\,,1) \in X((\vec{\sigma}\,,n))$. Thus

$$\mathrm{card}(X((\vec{\sigma}\,,n))) \ge 2\,\mathrm{card}(Y(\vec{\sigma}\,)) = 2^k,$$

contrary to our assumption, so our claim is established. Since

$$\mathrm{card}(Y(\vec{\sigma}\,)) < 2^{k-1}$$

whenever $\vec{\sigma}$ is an element of Σ^{n-1}_{k-1}, the induction hypothesis implies that

$$\mathrm{card}(Y) \le S(n-1,k-1).$$

Therefore

$$\begin{aligned}
\mathrm{card}(X) &= \mathrm{card}(X(1,\ldots,n-1)) + \mathrm{card}(Y)\\
&\le S(n-1,k) + S(n-1,k-1)\\
&= S(n,k)
\end{aligned}$$

because

$$\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1}.$$

∎

Lemma 4.8.10 *If* F_k^A *is computable by a set of* $2^k - 1$ *partial recursive functions, then* F_n^A *is computable by a set of* $S(n,k)$ *partial recursive functions, for every* $n \geq k$*.*

Proof: By assumption, there exist partial recursive functions $g_1, \ldots, g_{2^k-1}$ such that

$$(\forall q_1, \ldots, q_k)[\mathrm{F}_k^A(q_1, \ldots, q_k) \in \{g_i(q_1, \ldots, q_k) \mid 1 \leq i \leq 2^k - 1\}].$$

We say that a pair of k-tuples $(\vec{p}, \vec{w})$ is *consistent* if $\vec{w} \in \{g_i(\vec{p}) \mid 1 \leq i \leq 2^k - 1\}$. We say that a pair of n-tuples $(\vec{q}, \vec{x})$ is consistent if $(\vec{q}(\vec{\sigma}), \vec{x}(\vec{\sigma}))$ is consistent for all $\vec{\sigma}$ in Σ_k^n. Since F_n^A is computed by the set of functions $\{g_1, \ldots, g_{2^k-1}\}$, the pair $(\vec{q}, \mathrm{F}_n^A(\vec{q}))$ is consistent for all $\vec{q}$.

We will complete the proof by defining partial recursive functions $g_1', g_2', \ldots$ such that $g_i'(\vec{q})$ is the ith n-tuple $\vec{x}$ such that $(\vec{q}, \vec{x})$ is consistent. First we define the set T as the set of consistent pairs of n-tuples. Let

$$T = \{(\vec{q}, \vec{x}) \mid (\forall \vec{\sigma} \in \Sigma_k^n)[\vec{x}(\vec{\sigma}) \in \{g_i(\vec{q}(\vec{\sigma})) \mid 1 \leq i \leq 2^k - 1\}]\}.$$

The set T is r.e., because we are quantifying over a finite set and each function g_i is partial recursive. Let

$$X_{\vec{q}} = \{\vec{x} \mid (\vec{q}, \vec{x}) \in T\}.$$

Thus $X_{\vec{q}}$ is the set of vectors $\vec{x}$ such that $(\vec{q}, \vec{x})$ is consistent. Therefore $\mathrm{F}_n^A(\vec{q}) \in X_{\vec{q}}$, and for all $\vec{\sigma}$ in Σ_k^n

$$X_{\vec{q}}(\vec{\sigma}) \subseteq \{g_i(\vec{q}(\vec{\sigma})) \mid 1 \leq i \leq 2^k - 1\},$$

so $\mathrm{card}(X_{\vec{q}}(\vec{\sigma})) < 2^k$. By Lemma 4.8.10, $\mathrm{card}(X_{\vec{q}}) \leq S(n,k)$.

Since T is r.e., let M be a Turing machine that enumerates T without repetition. We compute $g_i'(\vec{q})$ as follows: simulate M until M has enumerated i pairs of the form $(\vec{q}, \vec{x})$; output the second element of the ith such pair. Thus

$$X_{\vec{q}} = \{g_i'(\vec{q}) \mid i \geq 1\}.$$

Since $\mathrm{card}(X_{\vec{q}}) \leq S(n,k)$ and the sequence $g_1'(\vec{q}), g_2'(\vec{q}), \ldots$ contains no repetitions,

$$X_{\vec{q}} = \{g_i'(\vec{q}) \mid 1 \leq i \leq S(n,k)\}.$$

Since $\mathrm{F}_n^A(\vec{q}) \in X_{\vec{q}}$, the function F_n^A is computed by the $S(n,k)$ partial recursive functions $g_1', \ldots, g_{S(n,k)}'$. ▮

Theorem 4.8.11 *If there exists a set B such that*

$$\mathrm{F}_k^A \in \mathrm{FQ}(k-1, B)$$

then there exists a set $C \equiv_{\mathrm{tt}} A$ such that

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}((k-2)\log n + O(1), C).$$

Proof: Assume that $\mathrm{F}_k^A \in \mathrm{FQ}(k-1, B)$. By Theorem 4.1.2(i), F_k^A is computable by a set of 2^{k-1} partial recursive functions. By Lemma 4.2.2, F_{k-1}^A is computable by a set of $2^{k-1}-1$ partial recursive functions. Therefore, by Lemma 4.8.10, F_n^A is computable by a set of $S(n, k-1)$ partial recursive functions. By Theorem 4.1.2(ii), for every $n \geq k$ there exists a set $C_n \in \mathrm{FQ}(1, \mathrm{F}_n^A)$ such that

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}(\lceil \log S(n, k-1) \rceil, C_n).$$

Let C_ω be the recursive join of $C_k, C_{k+1}, \ldots$. Then, for all $n \geq k$

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}(\lceil \log S(n, k-1) \rceil, C_\omega).$$

Since $S(n, k-1) = O(n^{k-2})$,

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}((k-2)\log n + O(1), C_\omega).$$

Let $C = A$ join C_ω, so that $C \leq_{\mathrm{tt}} A \leq_{\text{1-tt}} C$ and

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}((k-2)\log n + O(1), C).$$

▮

Corollary 4.8.12 *If A is not superterse then there exists a natural number k such that A is k-semiverbose.*

Proof: This follows directly from Theorem 4.8.11. ▮

Theorem 4.8.11 provides an alternative way of proving Theorem 4.6.4.

Corollary 4.8.13 *There is a superterse set in every nonrecursive truth-table degree.*

Proof: Let A be a nonrecursive set. If A is not superterse, then by Theorem 4.8.11 there exists a set $B \equiv_{\rm tt} A$ such that

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}(O(\log n), B).$$

If B is not superterse then there exists a set C such that

$$\mathrm{F}_n^B \in \mathrm{FQ}_{||}(O(\log n), C),$$

so

$$\mathrm{F}_{\log n}^B \in \mathrm{FQ}_{||}(O(\log\log n), C).$$

Therefore

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}(O(\log\log n), C),$$

which contradicts the Nonspeedup Theorem. Thus either A is superterse, or else B is a superterse set that is truth-table equivalent to A. ∎

Corollary 4.8.14 *If $A \in \mathrm{Q}(k, B)$ and B is not superterse, then A is not superterse.*

Proof: Assume that B is not superterse.

$$\begin{aligned}
\mathrm{F}_n^A &\in \underbrace{\mathrm{Q}(k,B) \,\|\, \cdots \,\|\, \mathrm{Q}(k,B)}_{n} && \text{because } A \in \mathrm{Q}(k,B) \\
&\subseteq \underbrace{\mathrm{FQ}(k,B) \,\|\, \cdots \,\|\, \mathrm{FQ}(k,B)}_{n} \\
&\subseteq \underbrace{\mathrm{FQ}_{||}(n,B) \circ \cdots \circ \mathrm{FQ}_{||}(n,B)}_{k} && \text{by Observation 2.2.17} \\
&= \underbrace{\mathrm{FQ}(1,\mathrm{F}_n^B) \circ \cdots \circ \mathrm{FQ}(1,\mathrm{F}_n^B)}_{k}.
\end{aligned}$$

Since B is not superterse, there exists a set C such that

$$\mathrm{F}_n^B \in \mathrm{FQ}_{||}(O(\log n), C),$$

by Theorem 4.8.11. Therefore,

$$\begin{aligned} \mathrm{F}_n^A &\in \underbrace{\mathrm{FQ}_{||}(O(\log n), C) \circ \cdots \circ \mathrm{FQ}_{||}(O(\log n), C)}_{k} \\ &\subseteq \mathrm{FQ}(O(k \log n), C). \end{aligned}$$

Therefore, A is not superterse. ∎

4.9 Decision Problems and Superterseness

Definition 4.9.1 A set B is *self-encoding* if

$$(\exists k)(\forall n)[\mathrm{Q}_{||}(n, B) \subseteq \mathrm{Q}(k, B)].$$

Theorem 4.9.2 *If the set B is self-encoding then B is either superterse or recursive.*

Proof: Suppose that B is self-encoding, so that for some k

$$(\forall n)[\mathrm{Q}_{||}(n, B) \subseteq \mathrm{Q}(k, B)].$$

Suppose also that B is not superterse, so that for some set A and some positive integer j

$$\mathrm{F}_j^B \in \mathrm{FQ}(j-1, A).$$

By Theorem 4.1.2(i), the function F_j^B is computable by a set of $2^{j-1} \leq 2^j - 1$ partial recursive functions. Therefore, by Lemma 4.8.10, F_n^B is computable by a set of $S(n, j)$ partial recursive functions. Since $S(n, j) = O(n^{j-1})$, choose n large enough so that $S(n, j) \leq n^j$. Thus F_n^B is computable by a set of n^j partial recursive functions. By Theorem 4.1.2(ii), there exists a set C in $\mathrm{Q}(1, \mathrm{F}_n^B)$ such that

$$\begin{aligned} \mathrm{F}_n^B &\in \mathrm{FQ}_{||}(\lceil j \log n \rceil, C) \\ &\subseteq \mathrm{FREC} \circ \left(\underbrace{\mathrm{Q}(1, \mathrm{F}_n^B) \parallel \cdots \parallel \mathrm{Q}(1, \mathrm{F}_n^B)}_{\lceil j \log n \rceil} \right) \qquad \text{because } C \in \mathrm{Q}(1, \mathrm{F}_n^B) \end{aligned}$$

$$
\begin{aligned}
&= \mathrm{FREC} \circ \left(\underbrace{\mathrm{Q}_{||}(n,B) \parallel \cdots \parallel \mathrm{Q}_{||}(n,B)}_{\lceil j \log n \rceil} \right) \\
&\subseteq \mathrm{FREC} \circ \left(\underbrace{\mathrm{Q}(k,B) \parallel \cdots \parallel \mathrm{Q}(k,B)}_{\lceil j \log n \rceil} \right) \qquad \text{because } \mathrm{Q}_{||}(n,B) \subseteq \mathrm{Q}(k,B) \\
&\subseteq \mathrm{FREC} \circ \left(\underbrace{\mathrm{FQ}_{||}(\lceil j \log n \rceil, B) \circ \cdots \circ \mathrm{FQ}_{||}(\lceil j \log n \rceil, B)}_{k} \right) \qquad \text{by Observation 2.2.17.}
\end{aligned}
$$

For all n, the function F_n^B is computable by a set of $S(n,j)$ partial recursive functions, as shown above. Since $S(n,j) = O(n^{j-1})$, choose n large enough so that $S(\lceil j \log n \rceil, j) \leq (\log n)^j$. Therefore there is a set C' such that

$$\mathrm{FQ}_{||}(\lceil j \log n \rceil, B) \subseteq \mathrm{FQ}_{||}(\lceil j \log \log n \rceil, C'),$$

by Theorem 4.1.2(ii). Therefore,

$$
\begin{aligned}
\mathrm{F}_n^B &\in \mathrm{FREC} \circ \underbrace{\mathrm{FQ}_{||}(\lceil j \log \log n \rceil, C') \circ \cdots \circ \mathrm{FQ}_{||}(\lceil j \log \log n \rceil, C')}_{k} \\
&\subseteq \mathrm{FQ}(k \lceil j \log \log n \rceil, C').
\end{aligned}
$$

Choose n large enough so that $n \geq 2^{k \lceil j \log \log n \rceil}$. By the Nonspeedup Theorem, B must be recursive. ▮

Corollary 4.9.3 *Let A be nonrecursive. If B is 1-query complete for $\mathrm{Q}_{||}(\omega, A)$ then B is superterse.*

Proof: Let B be 1-query complete for $\mathrm{Q}_{||}(\omega, A)$. Then $B \in \mathrm{Q}_{||}(\omega, A)$ and $\mathrm{Q}_{||}(\omega, A) \subseteq \mathrm{Q}(1, B)$. For every n,

$$
\begin{aligned}
\mathrm{Q}_{||}(n,B) &\in \mathrm{Q}_{||}(\omega, A) \qquad \text{because } B \in \mathrm{Q}_{||}(\omega, A) \\
&\subseteq \mathrm{Q}(1,B).
\end{aligned}
$$

Thus B is self-encoding. By Theorem 4.9.2, B is superterse or recursive. Because $A \in \mathrm{Q}(1,B)$, the set B is not recursive. Therefore B is superterse. ▮

Corollary 4.9.4 *If there exists k such that no 0,1-valued partial function belongs to*

$$\mathrm{FQ}_{||}(2k, B) - \mathrm{FQ}_{||}(k, B),$$

then B is superterse or recursive.

Proof: Assume that there is no 0,1-valued partial function in $\mathrm{FQ}_{||}(2k, B) - \mathrm{FQ}_{||}(k, B)$. By Theorem 4.4.4(ii), there is no 0,1-valued partial function in $\mathrm{FQ}_{||}(n, B) - \mathrm{FQ}_{||}(k, B)$ for any n. Therefore, there is no 0,1-valued total function in $\mathrm{FQ}_{||}(n, B) - \mathrm{FQ}_{||}(k, B)$. Therefore, for every n,

$$\mathrm{Q}_{||}(n, B) \subseteq \mathrm{Q}_{||}(k, B) \subseteq \mathrm{Q}(k, B),$$

so B is self-encoding. By Theorem 4.9.2, B is superterse or recursive. ∎

Corollary 4.9.5 *Let B be a nonrecursive set in $\mathrm{Q}(j, A)$, where A is not superterse. Then for every k there is a 0,1-valued partial function in*

$$\mathrm{Q}_{||}(2k, B) - \mathrm{Q}_{||}(k, B).$$

Proof: By Corollary 4.8.14, B is not superterse. Since B is not recursive either, there must be a 0,1-valued partial function in $\mathrm{Q}_{||}(2k, B) - \mathrm{Q}_{||}(k, B)$, by the preceding corollary. ∎

4.10 Discussion and Related Work

Since the appearance of the original draft of this chapter, several papers and technical reports have been published on the topic of bounded queries to a nonrecursive set [BGGO93, BG87, Bei87a, BGO89, Owi89, Bei87d].

Material from Sections 4.1, 4.2, 4.5, and 4.6 has been included in [BGGO93] by Beigel, Gasarch, Gill, and Owings. In [BGGO93], it was shown that all semirecursive sets are verbose. Using similar techniques, we can construct sets that are $(k+1)$-semiverbose but not k-semiverbose [BG].

Material from Section 4.4 has been included in [BG87], in which we defined supportive and parallel supportive sets. A set B is supportive if $\mathrm{Q}(n,B) \subset \mathrm{Q}(n+1,B)$ for all n. A set B is parallel supportive if $\mathrm{Q}_{||}(n,B) \subset \mathrm{Q}_{||}(n+1,B)$ for all n. In [BG87], we showed that the jump of every set is supportive and parallel supportive, all generic sets are supportive and parallel supportive, all semirecursive sets are supportive and parallel supportive, every truth-table degree contains a set that is supportive and parallel supportive, and every r.e. Turing degree contains an r.e. set that is supportive and parallel supportive. We also showed that the jump of every Turing degree contains a set that is not parallel supportive.

In [Bei87d], we showed that almost all sets are supportive and parallel supportive, and that all nonrecursive r.e. sets are supportive and parallel supportive. We constructed a set that is neither supportive nor parallel supportive.

Our paper [Bei87a] consists of the material from Section 4.3.

In [BGO89], Beigel, Gasarch, and Owings define bounded query classes for nondeterministic computation, and we study nondeterministic terseness.

Chapter 5

Polynomial Time Bounded Reductions

In the preceding chapters we considered machines that could perform arbitrary effective computations. In this chapter, we restrict our attention to machines that run in polynomial time. We define the bounded query classes for polynomial time bounded computations, and we attempt to generalize the results from the preceding chapters to polynomial time bounded computations. In Section 5.2, we prove a weak analogue of the Nonspeedup Theorem. In Section 5.7, we use the Weak Nonspeedup Theorem to show that that $k+1$ queries to an NP-hard oracle allow us to compute more functions in polynomial time than only k queries to the same oracle allow us to compute in polynomial time, unless $\mathsf{P} = \mathsf{NP}$. In Section 5.4, we discover that the Nonspeedup Theorem does not generalize to polynomial time bounded computations [AG88], and we study the sets for which the generalization fails. In Section 5.6, we prove a generalization of Theorem 4.9.2. In Section 5.7 we use that theorem to show that Δ_2^{P}-hard sets are polynomial superterse and to produce a relativization that makes all NP-hard sets polynomial superterse.

We define the bounded query classes for polynomial time:

- $\mathrm{MQ}(n, A, \mathsf{P})$ is the set of machines with oracle A that run in polynomial time and make at most n queries to A.

- $\mathrm{FQ}(n, A, \mathsf{P})$ is the set of total functions that are computable by a machine in $\mathrm{MQ}(n, A, \mathsf{P})$.
- $\mathrm{Q}(n, A, \mathsf{P})$ is the set of 0,1-valued total functions that are in $\mathrm{FQ}(n, A, \mathsf{P})$.
- $\mathrm{MQ}_{||}(n, A, \mathsf{P})$ is the set of machines with oracle A that run in polynomial time and make at most n queries to A, all queries being made in parallel.
- $\mathrm{FQ}_{||}(n, A, \mathsf{P})$ is the set of total functions that are computable by a machine in $\mathrm{MQ}_{||}(n, A, \mathsf{P})$.
- $\mathrm{Q}_{||}(n, A, \mathsf{P})$ is the set of 0,1-valued total functions that are in $\mathrm{FQ}_{||}(n, A, \mathsf{P})$.

5.1 Computability by a Set of Polynomial Time Functions

The material in this section is analogous to the material in Section 4.1.

Definition 5.1.1 The total function h is *computable by a set of k polynomial time functions* if there exist k polynomial time functions $g_1, \ldots, g_k$ such that

$$(\forall x)[h(x) \in \{g_i(x) \mid 1 \leq i \leq k\}].$$

Thus, the function h is computable by a set of k polynomial time functions if, for each x, we can compute in polynomial time a length-k list that includes $h(x)$. Informally, we say that *there are only k possible values* for $h(x)$.

Theorem 5.1.2

i. If there exists an oracle B such that $h \in \mathrm{FQ}(k, B, \mathsf{P})$ then h is computable by a set of 2^k polynomial time functions.

ii. If h is computable by a set of 2^k polynomial time functions then there exists an oracle $B \in \mathrm{Q}(1, h, \mathsf{P})$ such that $h \in \mathrm{FQ}_{||}(k, B, \mathsf{P})$.

Proof: The constructions in the proof of Theorem 4.1.2 run in polynomial time. ∎

This theorem enables us to show that every function h computable in polynomial time by making n serial queries to an oracle A can be computed in polynomial time by making n *parallel* queries to a different oracle B such that $B \in \mathsf{Q}(1, h, \mathsf{P})$.

Corollary 5.1.3 *If h is in $\mathrm{FQ}(n, A, \mathsf{P})$ then there exists a set B in $\mathsf{Q}(1, h, \mathsf{P})$ such that h is in $\mathrm{FQ}_{||}(n, B, \mathsf{P})$.*

Proof: By Theorem 5.1.2(i), h is computable by a set of 2^n polynomial time functions. Therefore, by Theorem 5.1.2(ii), there is a set B in $\mathsf{Q}(1, h, \mathsf{P})$ such that h is in $\mathrm{FQ}_{||}(n, B, \mathsf{P})$. ∎

5.2 A Weak Nonspeedup Theorem

In [AG88], Amir and Gasarch have shown how to produce a set B of arbitrarily large time complexity such that $\mathrm{F}_n^B \in \mathrm{FQ}(1, B, \mathsf{P})$ for every n. Thus the Nonspeedup Theorem does not apply to polynomial time computation; however, we can prove a weak version of the Nonspeedup Theorem for polynomial time computation. In Section 5.7, we will use the Weak Nonspeedup Theorem to prove that $\mathrm{FQ}_{||}(n, B, \mathsf{P}) \subset \mathrm{FQ}_{||}(n+1, B, \mathsf{P})$ for every NP-hard set B and every n, unless $\mathsf{P} = \mathsf{NP}$.

Definition 5.2.1 If $\mathcal{C}$ is a collection of sets and X is a set, then X *separates* $\mathcal{C}$ if for all S, S' in $\mathcal{C}$

$$S \neq S' \Rightarrow S \cap X \neq S' \cap X.$$

This section's main result will follow from the following combinatorial lemma, which says that $k-1$ points are sufficient to separate k sets. The lemma, which appears in [Owi89] was first stated and proved by Owings [Owi86]. We present Owings's proof.

Lemma 5.2.2 *If $|\mathcal{C}| = k \geq 1$ then there exists a set X that separates $\mathcal{C}$ such that $|X| \leq k-1$.*

Proof: By induction on k. The base case ($k = 1$) is trivial. Assume that the lemma is true for some value of $k \geq 1$. Let S_1, S_2 be distinct elements of $\mathcal{C}$, and let $x \in (S_1 - S_2) \cup (S_2 - S_1)$. Let

$$\mathcal{C}_1 = \{S \in \mathcal{C} \mid x \in S\} \text{ and } \mathcal{C}_2 = \{S \in \mathcal{C} \mid x \notin S\}.$$

Let $k_1 = |\mathcal{C}_1|$ and $k_2 = |\mathcal{C}_2|$. By the induction hypothesis, there exists a set X_1 that separates $\mathcal{C}_1$ such that $|X_1| \leq k_1 - 1$, and there exists a set X_2 that separates $\mathcal{C}_2$ such that $|X_2| \leq k_2 - 1$. Let $X = X_1 \cup X_2 \cup \{x\}$. Then X separates $\mathcal{C}$ and $|X| \leq k_1 - 1 + k_2 - 1 + 1 = k_1 + k_2 - 1 = k - 1$. ∎

We also present a different proof of Lemma 5.2.2, which is based on our original proof of the Weak Nonspeedup Theorem in [Bei86]. In our proof we first show that there is a finite set Y of m points that separates $\mathcal{C}$. We construct X by taking two cases: If there is one point that is "necessary" in order to separate two of the sets in $\mathcal{C}$ then we put the necessary point in X. If none of the points is necessary in order to separate any two of the sets in $\mathcal{C}$ then we remove any point from Y. In either case, we reduce to a smaller problem and proceed inductively.

Proof: We write $A \bigtriangleup B$ to denote $(A - B) \cup (B - A)$, the symmetric difference of A and B. If $x \in A \bigtriangleup B$ then we say that the point x separates A from B, and we say that the sets A and B differ on x.

For each pair of sets $S, S' \in \mathcal{C}$, we can choose a single point that separates S from S'; thus there exists a set Y that separates $\mathcal{C}$ such that $|Y| \leq \binom{k}{2}$. Therefore, it suffices to show that if $|\mathcal{C}| = k$ and a finite set separates $\mathcal{C}$ then there is a set X that separates $\mathcal{C}$ such that $|X| \leq k - 1$.

We prove that statement by induction on k. The base case ($k = 1$) is trivial. Assume that the statement is true for some value of $k \geq 1$, and let $\mathcal{C}$ be a collection of $k + 1$ distinct sets.

We prove, by induction on m, that if Y separates $\mathcal{C}$ and $|Y| = m$ then there exists a set X that separates $\mathcal{C}$ such that $|X| \leq k$. The base case ($m = 1$) is trivial because $k \geq 1$. Suppose that the statement is true for some $m \geq 1$, suppose that Y separates $\mathcal{C}$, and suppose that $|Y| = m + 1$.

Let $\mathcal{D} = \{S \cap Y \mid S \in \mathcal{C}\}$. We claim that if X separates $\mathcal{D}$, then X separates $\mathcal{C}$. Proof: Assume that X separates T. Let S and S' be distinct elements of $\mathcal{C}$. Then $S \cap Y \neq S' \cap Y$ because Y separates $\mathcal{C}$. Therefore $S \cap Y \cap X \neq S' \cap Y \cap X$, because X separates $\mathcal{D}$. Therefore $S \cap X \neq S' \cap X$, proving the claim. Thus it suffices to find a set X that separates $\mathcal{D}$ such that $|X| = k$. We consider two cases.

Case 1: *There exist two sets T, T' in $\mathcal{D}$ that differ on exactly one point.*

Let x be the unique element of $T \,\Delta\, T'$. There cannot be three sets that differ only on the single point x. However, there may be other pairs of sets in $\mathcal{D}$ that differ only on the point x. Let there be p such pairs including (T, T'). Let $\mathcal{D}_1$ consist of one element from each such pair. Let $\mathcal{D}_2$ consist of the other element from each such pair. Let $\mathcal{D}_3$ consist of the remaining elements of $\mathcal{D}$. Then $|\mathcal{D}_1| = |\mathcal{D}_2| = p$, and $|\mathcal{D}_3| = k - 2p + 1$.

By the induction hypothesis (for k) there exists a set X_0 that separates $\{T - \{x\} \mid T \in \mathcal{D}_1 \cup \mathcal{D}_3\}$ such that $|X_0| = p + (k - 2p + 1) - 1 = k - p$. Let $X = X_0 \cup \{x\}$. We claim that X separates $\mathcal{D}$. Proof: Let T, T' be distinct elements of $\mathcal{D}$. We take three cases.

Case (a): *T and T' are both in $\mathcal{D}_1 \cup \mathcal{D}_3$.*

By our choice of x, it follows that $T - \{x\} \neq T' - \{x\}$. We chose X_0 so that X_0 separates $T - \{x\}$ from $T' - \{x\}$. Therefore X_0 separates T from T'.

Case (b): *$T \in \mathcal{D}_1 \cup \mathcal{D}_3$ and $T' \in \mathcal{D}_2$.*

Let $T'' = T' \,\Delta \{x\}$. By construction, the set T'' is in $\mathcal{D}_1$. If $T'' = T$, then x separates T from T'. Otherwise, X_0 separates $T - \{x\}$ from $T'' - \{x\}$. Since $T'' - \{x\} = T' - \{x\}$, the set X_0 separates $T - \{x\}$ from $T' - \{x\}$. Therefore X_0 separates T from T'.

Case (c): *T and T' are both in $\mathcal{D}_2$.*

Let $T'' = T \,\Delta \{x\}$, and let $T''' = T' \,\Delta \{x\}$. Then T'' and T''' are distinct elements of $\mathcal{D}_1$. Therefore X_0 separates $T'' - \{x\}$ from $T''' - \{x\}$, so X_0 separates $T - \{x\}$ from $T' - \{x\}$. Therefore, X_0 separates T from T'.

In each case, $X = X_0 \cup \{x\}$ separates T from T'. Therefore X separates $\mathcal{D}$. Since $|X| = k - p + 1 \leq k$, the claim is proven.

Case 2: *Every pair of sets T, T' in $\mathcal{D}$ differs on at least two points.*
Each T in $\mathcal{D}$ is a subset of Y. Because $|Y| = m + 1$, any set of m points in Y separates $\mathcal{D}$. By the induction hypothesis (for m) there exists a set X that separates T such that $|X| \leq k$.

∎

Lemma 5.2.3 *If F_k^A is computable by a set of k polynomial time functions, then any k queries to A can be answered by a polynomial time algorithm that asks only $k-1$ of the same queries in parallel.*

Proof: By assumption, there exist k polynomial time functions $g_1, \ldots, g_k$ such that

$$(\forall x_1, \ldots, x_k)[\mathrm{F}_k^A(x_1, \ldots, x_k) \in \{g_i(x_1, \ldots, x_k) \mid 1 \leq i \leq k\}].$$

Without loss of generality, assume that if $i \neq j$ then $g_i(x_1, \ldots, x_k) \neq g_j(x_1, \ldots, x_k)$ for all $x_1, \ldots, x_k$. Let $\vec{x} = (x_1, \ldots, x_k)$ and let $X = \{x_1, \ldots, x_k\}$. For $i = 1, \ldots, k$ let

$$S_i = \{x \in X \mid g_i(x) = 1\} \subseteq X$$

We say that the set S agrees with the set A on X if $S \cap X = A \cap X$. Because F_n^A is computed by $g_1, \ldots, g_k$, one of the sets $S_1, \ldots, S_k$ agrees with A on X. Thus we can determine $\mathrm{F}_k^A(\vec{x}\,)$ by computing a natural number i such that S_i agrees with A on X.

Because the functions $g_1, \ldots, g_k$ produce distinct outputs, the sets $S_1, \ldots, S_k$ are distinct. By Lemma 5.2.2, there is a $(k-1)$-element set $X' = \{x'_1, \ldots, x'_{k-1}\}$ that separates $\{S_1, \ldots, S_k\}$. Since $S_1, \ldots, S_k$ are subsets of X, points outside of X cannot help to separate $\{S_1, \ldots, S_k\}$; therefore, without loss of generality, we may assume that X' is a subset of X. Because one of the sets $S_1, \ldots, S_k$ agrees with A on X, at least one of the sets $S_1, \ldots, S_k$ agrees with A on X'. Because X' separates $\{S_1, \ldots, S_k\}$, exactly one of the sets $S_1, \ldots, S_k$ agrees with A on X'. This set must also agree with

A on X. Thus we can determine $\mathrm{F}_k^A(\vec{x}\,)$ by computing the unique i such that S_i agrees with A on X'.

The following algorithm computes F_k^A:

Step 1: Input $x_1, \ldots, x_k$.

Step 2: Compute $S_1, \ldots, S_k$ as above.

Step 3: By using the construction implicit in Lemma 5.2.2 ($*$ or by trying all k possibilities $*$) find a set of $k-1$ points $\{x'_1, \ldots, x'_{k-1}\} \subset \{x_1, \ldots, x_k\}$ that separates $\{S_1, \ldots, S_k\}$.

Step 4: Compute $\mathrm{F}_{k-1}^A(x'_1, \ldots, x'_{k-1})$.

Step 5: Find i such that

$$\mathrm{F}_{k-1}^A(x'_1, \ldots, x'_{k-1}) = \mathrm{F}_{k-1}^{S_i}(x'_1, \ldots, x'_{k-1}).$$

Step 6: Output $g_i(x_1, \ldots, x_k)$.

∎

Theorem 5.2.4 *If* $\mathrm{F}_{2^k}^A \in \mathrm{FQ}(k, B, \mathsf{P})$ *then*

i. for every $n \geq 2^k$*, any* n *queries to* A *can be answered by a polynomial time algorithm that asks only* $2^k - 1$ *of the same queries in parallel.*

ii. for every n*,* $\mathrm{F}_n^A \in \mathrm{FQ}(k, B, \mathsf{P})$*.*

Proof: Assume that $\mathrm{F}_{2^k}^A \in \mathrm{FQ}(k, B, \mathsf{P})$.

i. By Theorem 5.1.2(i), $\mathrm{F}_{2^k}^A$ is computable by a set of 2^k polynomial time functions. Thus by Lemma 5.2.3, the answers to 2^k parallel queries to A can be determined in polynomial time by making (in parallel) only $2^k - 1$ of the same $2^k - 1$ queries to A.

   If $n > 2^k$ then we can replace 2^k of the n parallel queries with only $2^k - 1$ of them, thereby eliminating one of the n queries. We keep eliminating queries in this way until we are left with only $2^k - 1$ of the original n parallel queries.

ii. This is obvious if $n < 2^k$. If $n \geq 2^k$ then

$$\begin{aligned} \mathrm{F}_n^A &\subseteq \mathrm{FQ}_{||}(2^k - 1, A, \mathsf{P}) && \text{by (i)} \\ &\subseteq \mathrm{FQ}_{||}(2^k, A, \mathsf{P}) \\ &\subseteq \mathrm{FQ}(k, B, \mathsf{P}) && \text{by assumption.} \end{aligned}$$

▮

5.3 A Serial-Parallel Tradeoff

In this section, we generalize Lemma 3.2.1, which states that

$$\mathrm{FQ}(n, K) \subseteq \mathrm{FQ}_{||}(2^n - 1, K).$$

In the recursion theoretic framework of the preceding chapters, that result is not true for all oracles. However, in the polynomial time bounded framework of the current chapter, that result is true for all oracles.

A well-known theorem of Nerode [Rog67, Theorem 9-XIX] states that if A is Turing reducible to B by a reduction that terminates regardless of the oracle answers, then A is truth-table reducible to B. Because polynomial time reductions always terminate, we obtain a similar result for polynomial time bounded query reductions.

Theorem 5.3.1 *For every set A and natural number k,*

$$\mathrm{FQ}(k, A, \mathsf{P}) \subseteq \mathrm{FQ}_{||}(2^k - 1, A, \mathsf{P}).$$

Proof: Let $f \in \mathrm{FQ}(k, A, \mathsf{P})$ and let f be computable in time $p(n)$ for some polynomial p. We can simulate the computation of f for all possible sequences of oracle answers in time $O(2^k p(n))$, because we can truncate any computation that runs for more than $p(n)$ steps. During the simulation we prepare a list of all $2^k - 1$ possible queries. We make the $2^k - 1$ queries in parallel, and then simulate f with the correct sequence of oracle answers. ▮

Corollary 5.3.2 *If* $\mathrm{F}^A_{2^k} \in \mathrm{FQ}(k, B, \mathsf{P})$ *then for every* n

$$\mathrm{FQ}(n, A, \mathsf{P}) \subseteq \mathrm{FQ}(k, B, \mathsf{P}).$$

Proof: Assume that $\mathrm{F}^A_{2^k} \in \mathrm{FQ}(k, B, \mathsf{P})$.

$$\begin{aligned} \mathrm{FQ}(n, A, \mathsf{P}) &\subseteq \mathrm{FQ}_{||}(2^n - 1, A, \mathsf{P}) \qquad \text{by Theorem 5.3.1} \\ &\subseteq \mathrm{FQ}(k, B, \mathsf{P}) \qquad \text{by Theorem 5.2.4(ii).} \end{aligned}$$

▮

Corollary 5.3.3 *There exists a set* B *of arbitrarily great time complexity such that for every* k

$$\mathrm{FQ}(k, B, \mathsf{P}) \subseteq \mathrm{FQ}(1, B, \mathsf{P}).$$

Proof: In [AG88], Amir and Gasarch have shown how to construct a set B of arbitrarily great time complexity such that $\mathrm{F}^B_k \in \mathrm{FQ}(1, B, \mathsf{P})$ for every k. In particular $\mathrm{F}^B_{2^k-1} \in \mathrm{FQ}(1, B, \mathsf{P})$. Therefore, by Theorem 5.3.1

$$\mathrm{FQ}(k, B, \mathsf{P}) \subseteq \mathrm{FQ}_{||}(2^k - 1, B, \mathsf{P}) \subseteq \mathrm{FQ}(1, B, \mathsf{P}).$$

▮

5.4 Cheatable Sets

In this section, we study the class of sets for which the polynomial time version of the Nonspeedup Theorem fails.

Definition 5.4.1

- A set A is *k-cheatable* if $(\exists B)[F^A_{2^k} \in \mathrm{FQ}(k, B, \mathsf{P})]$.
- A set A is *cheatable* if A is k-cheatable for some k.

The name cheatable is motivated by Theorem 5.2.4(i), which states that if B is cheatable then any n queries to B can be answered by a polynomial time algorithm that asks only a fixed number (independent of n) of the same questions. If the answers to a true-false test are given by a cheatable set, then a student up to no good would only need to copy a fixed number of answers in order to determine them all.

As mentioned in Section 5.2, Amir and Gasarch have constructed 1-cheatable sets. The proof below is based on their proof in [AG88].

Theorem 5.4.2 (Amir and Gasarch) *There exists a 1-cheatable set that is not in* P*.*

Proof: Define $\mathrm{tow}(n)$ recursively as follows:

$$\mathrm{tow}(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2^{\mathrm{tow}(n-1)} & \text{if } n \geq 1. \end{cases}$$

By the time hierarchy theorem [HU79, Theorem 12.9, p. 299], there exists a set $L \subseteq 0^*$ that is in $\mathrm{DTIME}((\mathrm{tow}(n+1))^2)$, but not in $\mathrm{DTIME}(\mathrm{tow}(n+1))$. Let

$$A = \{0^{\mathrm{tow}(y)} \mid 0^y \in L\}.$$

If $A \in \mathrm{DTIME}(n^k)$, then $L \in \mathrm{DTIME}((\mathrm{tow}(n))^k) \subseteq \mathrm{DTIME}(\mathrm{tow}(n+1))$, contradicting our choice of L; therefore $A \notin \mathsf{P}$. Given $x_1 = \mathrm{tow}(y_1) < x_2 = \mathrm{tow}(y_2)$, the running time to determine if x_1 is in A is quadratic in the length of x_2. Therefore a single query to A (to determine $\chi_A(x_2)$) and a polynomial amount of running time (to determine $\chi_A(x_1)$) allow us to compute $\mathrm{F}_2^A(x_1, x_2)$. Therefore, A is 1-cheatable. ∎

Theorem 5.4.3

i. If $\mathrm{FQ}(k+1, B, \mathsf{P}) = \mathrm{FQ}(k, B, \mathsf{P})$ *then* B *is* k*-cheatable.*

ii. If $\mathrm{FQ}_{||}(k+1, B, \mathsf{P}) = \mathrm{FQ}_{||}(k, B, \mathsf{P})$ *then* B *is* k*-cheatable.*

Proof:

i. Using the same proof as of Observation 2.2.15(i), we see that

$$(\forall n \geq k)[\mathrm{FQ}(n, B, \mathsf{P}) = \mathrm{FQ}(k, B, \mathsf{P})].$$

In particular $\mathrm{FQ}(2^k, B, \mathsf{P}) = \mathrm{FQ}(k, B, \mathsf{P})$, so $\mathrm{F}^B_{2^k} \in \mathrm{FQ}(k, B, \mathsf{P})$.

ii. Using the same proof as of Observation 2.2.15(ii), we see that

$$(\forall n \geq k)[\mathrm{FQ}_{||}(n, B, \mathsf{P}) = \mathrm{FQ}_{||}(k, B, \mathsf{P})].$$

In particular $\mathrm{FQ}_{||}(2^k, B, \mathsf{P}) = \mathrm{FQ}_{||}(k, B, \mathsf{P})$, so

$$\mathrm{F}^B_{2^k} \in \mathrm{FQ}_{||}(k, B, \mathsf{P}) \subseteq \mathrm{FQ}(k, B, \mathsf{P}).$$

∎

Self-reducible sets were defined by Schnorr in [Sch76]:

Definition 5.4.4 A set B is *self-reducible* if there exists a polynomial time bounded oracle machine M such that for every string x the machine M^B determines whether x is in B by querying only strings that are shorter than x.

We say that a set B is *self-tt-reducible* if we can determine in polynomial time the answer to any query x to B by asking several parallel queries to B, all of which are shorter than x.

Definition 5.4.5 The set B is *self-tt-reducible* if there exist polynomial time computable functions f and $\vec{q}$ such that for every string x

$$\chi_B(x) = f(x, \mathrm{F}_\omega(\vec{q}(x))),$$

and each component of $\vec{q}(x)$ is shorter than x.

Theorem 5.4.6 *If B is self-tt-reducible and cheatable, then $B \in \mathsf{P}$.*

Proof: Since B is self-tt-reducible, there exist polynomial time computable functions f and $\vec{q}$ such that for every string x

$$\chi_B(x) = f(x, \mathrm{F}_\omega(\vec{q}\,(x))),$$

and each component of $\vec{q}\,(x)$ is shorter than x. Assume that B is k-cheatable. The following recursive algorithm computes $\mathrm{F}^B_{2^k-1}$:

Step 1: Input $\vec{x} = (x_1, \ldots, x_{2^k-1})$.

Step 2: If each component of $\vec{x}$ is equal to the empty string, then compute $\mathrm{F}^B_{2^k-1}(\vec{x}\,)$ by table lookup, and return the value.

Step 3: Compute $\vec{y}$ by concatenating $\vec{q}\,(x_1), \ldots, \vec{q}\,(x_{2^k-1})$. If the length of $\vec{y}$ is less than 2^k then pad $\vec{y}$ with empty strings, so that the length of $\vec{y}$ is 2^k.

Step 4: As in the proof of Theorem 5.2.4(i), we can compute $\mathrm{F}^B_\omega(\vec{y}\,)$ in polynomial time by determining the answers to only $2^k - 1$ of the same queries. Let $\vec{z} = (z_1, \ldots, z_{2^k-1})$ be those queries. Recursively compute $\mathrm{F}^B_{2^k-1}(\vec{z}\,)$, and use the answer in order to compute $\mathrm{F}^B_\omega(\vec{y}\,)$.

Step 5: Return the value of $(f(x_1, \mathrm{F}^B_\omega(\vec{q}\,(x_1))), \ldots, f(x_{2^k-1}, \mathrm{F}^B_\omega(\vec{q}\,(x_{2^k-1}))))$.

Since each component of $\vec{q}\,(x_i)$ is shorter than x_i, the depth of the recursion is bounded by the length of the longest component of $\vec{x}$. Each recursive call runs in polynomial time, so the algorithm runs in polynomial time. Since $\mathrm{F}^B_{2^k-1}$ is computable in polynomial time, the set B is computable in polynomial time. ∎

Corollary 5.4.7 *If B is self-tt-reducible and $B \notin \mathsf{P}$ then*

i. $\mathrm{FQ}(k, B, \mathsf{P}) \subset \mathrm{FQ}(k+1, B, \mathsf{P})$.

ii. $\mathrm{FQ}_{||}(k, B, \mathsf{P}) \subset \mathrm{FQ}_{||}(k+1, B, \mathsf{P})$.

Proof: Let B be a self-tt-reducible set that is not in P.

i. By contradiction. Assume that $\mathrm{FQ}(k, B, \mathsf{P}) = \mathrm{FQ}(k+1, B, \mathsf{P})$. By Theorem 5.4.3, B is cheatable. By assumption, B is self-tt-reducible and $B \notin \mathsf{P}$. This contradicts Theorem 5.4.6.

ii. Similar to (i).

▮

By Theorem 5.4.6, every self-tt-reducible, cheatable set is in P. In Section 5.7, we will show that if $\mathsf{P} \neq \mathsf{NP}$ then no NP-hard set is cheatable. We would like to show that other classes of sets contain no cheatable sets; in particular, we would like to show that cheatable sets are, in some sense, easy. In the next theorem, we show that any 1-cheatable set must be easy infinitely often. In [BS85], Balcazar and Schöning formalized the notion of being easy infinitely often, which was previously considered by Berman and Hartmanis [BH77] and by Rabin [Rab60].

Definition 5.4.8 A set A is bi-immune for a class $\mathcal{C}$ if neither A nor $\bar{A}$ has an infinite subset that belongs to $\mathcal{C}$.

Thus A is easy infinitely often if and only if A is not bi-immune for P. Theorem 5.4.10 below shows that every 1-cheatable set is easy infinitely often.

Lemma 5.4.9 *If A is 1-cheatable then there is a polynomial time algorithm that takes as input two queries to A and either determines the answer to one of the queries or else determines whether the two answers are equal or unequal.*

Proof: Let A be a 1-cheatable set. By Theorem 5.2.4(i), we can answer any two queries x, y to A by asking only one of them. Let M be a polynomial time bounded machine that performs that computation.

The polynomial time algorithm below takes as input two queries x and y and produces one of the following six answers: "$\chi_A(x) = 1$," "$\chi_A(x) = 0$," "$\chi_A(y) = 1$," "$\chi_A(y) = 0$," "$\chi_A(x) = \chi_A(y)$," or "$\chi_A(x) \neq \chi_A(y)$."

Step 1: Input two queries x, y.

Step 2: Simulate M until M is about to make its query q; let p be the other query.

Step 3: Simulate M for both possible oracle answers, thus determining a Boolean function f such that $\chi_A(p) = f(\chi_A(q))$.

Step 4: If the Boolean function f is identically true then output that $\chi_A(p) = 1$. If f is identically false then output that $\chi_A(p) = 0$. If f is the identity function then output that $\chi_A(p) = \chi_A(q)$. If f is the complement of the identity function then output that $\chi_A(p) \neq \chi_A(q)$.

∎

Theorem 5.4.10 *No 1-cheatable set is bi-immune for* P.

Proof: By the Nonspeedup Theorem, A must be recursive, so let M be a machine that decides membership in A. Call the algorithm of the preceding lemma Algorithm $\mathcal{A}$.

Our proof will proceed as follows: We define infinitely many sparse, infinite, disjoint sets $S_1, S_2, \ldots$. We run Algorithm $\mathcal{A}$ on every pair of consecutive elements of S_i. Either we determine the answer to one query in one of the pairs, or else we produce a long chain of queries, such that we know the relationship between $\chi_A(x)$ and $\chi_A(y)$ for consecutive elements x and y of the chain. In the latter case, we compute the answer to the smallest element of the chain, thereby determining the answer to the largest element of the chain. In either case, at least one element of S_i is easy. Since the sets $S_1, S_2, \ldots$ are disjoint, there are infinitely many easy points.

Let $f_i(n) = 0^n 1^i$, and let $S_i = \{f_i(n) \mid n \geq 0\}$. The construction below produces two sets B_0^i and B_1^i such that the the set B_0^i is a subset of $\bar{A}$, the set B_1^i is a subset of A, and $|B_0^i \cup B_1^i| = 1$.

Stage 0: Let $base = f_i(0)$. Let $equal = 1$. Go to stage 1.

Stage $n \geq 1$: Let $x = f_i(n)$, let $y = f_i(n+1)$, and run Algorithm $\mathcal{A}$ on input x, y. We take six cases, depending on the output of $\mathcal{A}$.

$\chi_A(x) = 0$: Let $B_0^i = \{x\}$ and $B_1^i = \emptyset$. Exit the construction.

$\chi_A(x) = 1$: Let $B_1^i = \{x\}$ and $B_0^i = \emptyset$. Exit the construction.

$\chi_A(y) = 0$: Let $B_0^i = \{y\}$ and $B_1^i = \emptyset$. Exit the construction.

$\chi_A(y) = 1$: Let $B_1^i = \{y\}$ and $B_0^i = \emptyset$. Exit the construction.

$\chi_A(x) = \chi_A(y)$: Do nothing.

$\chi_A(x) \neq \chi_A(y)$: Let $equal = 1 - equal$.

If M does not converge within n steps on input $base$, then go to stage $n+1$. (* Otherwise, if $equal = 1$ then $\chi_A(y) = \chi_A(base)$, else $\chi_A(y) \neq \chi_A(base)$. *) If $\chi_A(base) = equal$ then let $B_1^i = \{y\}$ and $B_0^i = \emptyset$; otherwise, let $B_0^i = \{y\}$ and $B_1^i = \emptyset$. Exit the construction.

Let

$$B_0 = \bigcup_{i \geq 1} B_0^i \quad \text{and} \quad B_1 = \bigcup_{i \geq 1} B_1^i.$$

We can determine whether z is of the form $f_i(n)$ in polynomial time. If $z = f_i(n)$, then we can determine whether z is in B_0 or B_1 (or neither) by running the construction above through stage $n < |z|$. Algorithm $\mathcal{A}$ runs in polynomial time. The remainder of time in each stage is dominated by the simulation of n steps of a Turing machine computation. The simulation can be performed in $O(n^2)$ time [HU79, Theorem 12.5]. Thus $B_0 \in \mathsf{P}$ and $B_1 \in \mathsf{P}$.

For every i, the set $B_0^i \cup B_1^i$ is nonempty, so $B_0 \cup B_1$ is infinite. Therefore B_0 or B_1 is infinite. Thus B_0 is an infinite polynomial time subset of $\bar{A}$, or B_1 is an infinite polynomial time subset of A. ∎

Gasarch has found a simpler proof of our result [Gas87]. We present his proof.

Proof: We take two cases.

Case 1: *For infinitely many values of n, when we run Algorithm $\mathcal{A}$ on input $(0^n, 0^{n+1})$, the algorithm tells us $\chi_A(0^n)$ or $\chi_A(0^{n+1})$.* In this case, let

$$B_v = \{0^n \mid \text{Algorithm } \mathcal{A} \text{ on input } (0^{n-1}, 0^n) \text{ or } (0^n, 0^{n+1}) \text{ yields } \chi_A(0^n) = v\},$$

for $v = 0, 1$. Then B_0 is an infinite polynomial time subset of $\bar{A}$, or B_1 is an infinite polynomial time subset of A.

Case 2: *There exists m such that for every natural number $n \geq m$, when we run Algorithm $\mathcal{A}$ on input $(0^n, 0^{n+1})$, the algorithm tells us whether $\chi_A(0^n) = \chi_A(0^{n+1})$.* In this case, the following algorithm determines whether $0^n \in A$: If $n \leq m$ then determine the answer by table lookup. If $n > m$ then determine whether $0^m \in A$ by table lookup; run Algorithm $\mathcal{A}$ on $(0^m, 0^{m+1})$, on $(0^{m+1}, 0^{m+2})$, ..., and on $(0^{n-1}, 0^n)$; and determine whether $0^n \in A$ by following the chain of answers given by Algorithm $\mathcal{A}$.

Let $B_v = \{0^n \mid \chi_A(0^n) = v\}$, for $v = 0, 1$. Then B_0 is an infinite subset of $\bar{A}$, or B_1 is an infinite subset of A.

∎

Definition 5.4.11 A set S is P-countable if $S = \{g(i) \mid i \in \mathsf{N}\}$, where g is 1-1, and both g and g^{-1} are polynomial time computable.

Allender has proved that if S is P-printable [HY84] then S is sparse and P-countable [All86, Theorem 3(2)]. However, P-countable sets need not be sparse.

Definition 5.4.12 A set A is locally bi-immune for a class $\mathcal{C}$ if there exists a P-countable set S such that neither $S \cap A$ nor $S \cap \bar{A}$ has an infinite subset that belongs to $\mathcal{C}$.

Theorem 5.4.13 *No 1-cheatable set is locally bi-immune for* P.

Proof: Let A be a 1-cheatable set, let g be a 1-1 function such that g and g^{-1} are polynomial time computable, and let $S = \{g(i) \mid i \in \mathsf{N}\}$. We perform the same construction as in the proof of Theorem 5.4.10, except that we let $x = g(f_i(n))$ and $y = g(f_i(n+1))$ (so that we will construct subsets of S), and instead of simulating M for n steps on input *base* we simulate M for $|y|$ steps on input *base*. Because g and f_i are 1-1 functions, $y = g(f_i(n+1))$ is unbounded; therefore the modified construction converges. Either B_0 is an infinite subset of $S \cap \bar{A}$, or B_1 is an infinite subset of $S \cap A$.
∎

Thus, we have proved that every cheatable set is easy infinitely often on every P-countable set.

5.5 P-Terse and P-Verbose Sets

In this section, we generalize definitions from Chapter 4, such as terseness and verboseness, to polynomial time computation. We show that if a set A is not P-superterse then n parallel queries to A can be answered in polynomial time by making only $O(\log n)$ queries to some oracle.

Definition 5.5.1

- A set A is *polynomial terse* (P-terse) if

$$(\forall n)[\mathrm{F}_n^A \notin \mathrm{FQ}(n-1, A, \mathsf{P})].$$

- A set A is *polynomial superterse* (P-superterse) if

$$(\forall B)(\forall n)[\mathrm{F}_n^A \notin \mathrm{FQ}(n-1, B, \mathsf{P})].$$

- A set A is *polynomial verbose* (P-verbose) if

$$(\forall n)[\mathrm{F}_{2^n-1}^A \in \mathrm{FQ}(n, A, \mathsf{P})].$$

- A set A is *polynomial k-verbose* (P-k-verbose) if

$$\mathrm{F}_n^A \in \mathrm{FQ}(k \log n + O(1), A, \mathsf{P}).$$

- A set A is *polynomial k-semiverbose* (P-k-semiverbose) if there exists a set B such that

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}(k \log n + O(1), B, \mathsf{P}).$$

- A set A is *polynomial self-encoding* (P-self-encoding) if

$$(\exists k)(\forall n)[\mathrm{Q}_{||}(n, A, \mathsf{P}) \subseteq \mathrm{Q}(k, A, \mathsf{P})].$$

The definition of P-terseness is due to Amir and Gasarch [AG88].

The following lemma is a generalization of Lemma 4.8.10. We have to be slightly careful in order to produce functions that run in polynomial time; otherwise the proof is the same.

Lemma 5.5.2 *If* F_k^A *is computable by a set of* $2^k - 1$ *polynomial time functions, then* F_n^A *is computable by a set of* $S(n,k)$ *polynomial time functions, for every* $n \geq k$.

Proof: By assumption, there exist polynomial time functions $g_1, \ldots, g_{2^k-1}$ such that

$$(\forall q_1, \ldots, q_k)[\mathrm{F}_k^A(q_1, \ldots, q_k) \in \{g_i(q_1, \ldots, q_k) \mid 1 \leq i \leq 2^k - 1\}].$$

We say that a pair of k-tuples $(\vec{p}, \vec{w})$ is *consistent* if $\vec{w} \in \{g_i(\vec{p}) \mid 1 \leq i \leq 2^k - 1\}$. If $m > k$, we say that a pair of m-tuples $(\vec{q}, \vec{x})$ is consistent if $(\vec{q}(\vec{\sigma}), \vec{x}(\vec{\sigma}))$ is consistent for all $\vec{\sigma}$ in Σ_k^m. Since F_k^A is computed by the set of functions $\{g_1, \ldots, g_{2^k-1}\}$, the pair $(\vec{q}, \mathrm{F}_n^A(\vec{q}))$ is consistent for every n-tuple $\vec{q}$.

We will complete the proof by defining polynomial time functions $g_1', g_2', \ldots$ such that $g_i'(\vec{q})$ is the ith n-tuple $\vec{x}$ such that $(\vec{q}, \vec{x})$ is consistent.

We can test in polynomial time whether a pair of k-tuples is consistent, because each function g_i is polynomial time computable and $2^k - 1$ is a constant. If $m > k$, we can test in polynomial time whether a pair of m-tuples is consistent, because there are only $\binom{m}{k}$ choices for $\vec{\sigma}$. We define a function g_i', which is computed as follows:

Step 1: Input $\vec{q} = (q_1, \ldots, q_n)$.

Step 2: Compute a list L_k containing all k-tuples $(x_1, \ldots, x_k)$ such that $(\vec{q}(1, \ldots, k), (x_1, \ldots, x_k))$ is consistent.

Step 3: For $j = k + 1$ to n do the following:

(a) Let L_j be an empty list.

(b) For each element $\vec{x}$ of L_{j-1} and for $b = 0, 1$ do the following:
if $(\vec{q}(1, \ldots, j), (\vec{x}, b))$ is consistent then insert $(\vec{x}, b)$ into L_j.

Step 4: If the length of L_n is at least i then output the ith element of L_n; otherwise, output the first element of L_n.

Let X_j be the set of elements on the list L_j at the end of step 3(b). Then $(\vec{q}(1, \ldots, j), \vec{x})$ is consistent for every j-tuple $\vec{x}$ in X_j. Therefore, for every $\vec{\sigma}$ in Σ_k^j

$$X_j(\vec{\sigma}) \subseteq \{g_i((\vec{q}(1, \ldots, j))(\vec{\sigma})) \mid 1 \leq i \leq 2^k - 1\},$$

so $\text{card}(X_j(\vec{\sigma}\,)) \le 2^k - 1 < 2^k$. By Lemma 4.8.9, $\text{card}(X_j) \le S(j,k) \le S(n,k)$, which is a polynomial in n. Since L_j contains no repetitions, the length of L_j is is bounded by a polynomial in n. Therefore g'_i is computable in polynomial time. Since L_n contains every n-tuple $\vec{x}$ such that $(\vec{q}, \vec{x}\,)$ is consistent,

$$\mathrm{F}_n^A(\vec{q}\,) \in \{g'_i(\vec{q}\,) \mid i \ge 1\}.$$

The length of L_n is at most $S(n,k)$; therefore $g'_i(\vec{q}\,) = g'_1(\vec{q}\,)$ for every $i > S(n,k)$. Therefore,

$$\mathrm{F}_n^A(\vec{q}\,) \in \{g'_i(\vec{q}\,) \mid 1 \le i \le S(n,k)\}.$$

∎

Theorem 5.5.3 *If there exists a set B such that*

$$\mathrm{F}_k^A \in \mathrm{FQ}(k-1, B, \mathsf{P})$$

then there exists a set $C \equiv_{\mathrm{tt}}^{\mathsf{P}} A$ such that

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}((k-1)\log n + O(1), C, \mathsf{P}).$$

Proof: Assume that $\mathrm{F}_k^A \in \mathrm{FQ}(k-1, B, \mathsf{P})$. By Theorem 5.1.2(i), F_k^A is computable by a set of $2^{k-1} \le 2^k - 1$ polynomial time functions. Therefore, by Lemma 5.5.2, F_n^A is computable by a set of $S(n,k)$ polynomial time functions. By Theorem 5.1.2(ii), for every $n \ge k$ there exists a set $C_n \in \mathrm{Q}_{||}(1, \mathrm{F}_n^A, \mathsf{P})$ such that

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}(\lceil \log (S(n,k)) \rceil, C_n, \mathsf{P}).$$

A review of the proof shows that the polynomial time bounds are the same for $C_k, C_{k+1}, \ldots$. Let C_ω be the recursive join [Rog67, Soa87] of $C_k, C_{k+1}, \ldots$. Then C_ω is polynomial time truth-table reducible to A, and for all $n \ge k$

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}(\lceil \log (S(n,k)) \rceil, C_\omega, \mathsf{P}).$$

Since $S(n,k) = O(n^{k-1})$,

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}((k-1)\log n + O(1), C_\omega, \mathsf{P}).$$

Let $C = A$ join C_ω, so that $C \leq_{\mathrm{tt}}^{\mathsf{P}} A \leq_{1\text{-}\mathrm{tt}}^{\mathsf{P}} C$ and

$$\mathrm{F}_n^A \in \mathrm{FQ}_{||}((k-1)\log n + O(1), C, \mathsf{P}).$$

▮

Corollary 5.5.4 *If A is not P-superterse, then there exists a natural number k such that A is P-k-semiverbose.*

Proof: This follows immediately from Theorem 5.5.3. ▮

5.6 Decision Problems and P-terse Sets

Theorem 5.6.1 *If the set B is P-self-encoding then*

$$(\exists k)(\forall n)[\mathrm{Q}(n, B, \mathsf{P}) \subseteq \mathrm{Q}_{||}(k, B, \mathsf{P})].$$

Proof: Assume that B is P-self-encoding so that for some j

$$(\forall n)[\mathrm{Q}_{||}(n, B, \mathsf{P}) \subseteq \mathrm{Q}(j, B, \mathsf{P})]. \tag{9}$$

Let $k = 2^j - 1$. By Theorem 5.3.1,

$$\begin{aligned}
\mathrm{Q}(n, B, \mathsf{P}) &\subseteq \mathrm{Q}_{||}(2^n - 1, B, \mathsf{P}) && \text{by Theorem 5.3.1} \\
&\subseteq \mathrm{Q}(j, B, \mathsf{P}) && \text{by equation (9)} \\
&\subseteq \mathrm{Q}_{||}(2^j - 1, B, \mathsf{P}) && \text{by Theorem 5.3.1} \\
&= \mathrm{Q}_{||}(k, B, \mathsf{P}).
\end{aligned}$$

▮

If the set B is P-self-encoding, then extra queries to B do not allow us to solve extra decision problems. The next theorem shows that if B is P-self-encoding but not P-superterse, then extra queries to B do not even allow us to compute extra functions.

Theorem 5.6.2 *If the set B is P-self-encoding but not P-superterse, then B is cheatable.*

Proof: The proof is the same as the proof of Theorem 4.9.2, except that we omit the last step, which applies the Nonspeedup Theorem. ▮

5.7 NP-Hard and Δ_2^{P}-Hard Oracles

In this section, we apply the results of the previous sections to NP-hard and Δ_2^{P}-hard oracles. We show that $n+1$ queries to an NP-hard oracle allow us to compute in polynomial time more functions than we can compute in polynomial time with only n queries to the same oracle, unless $\mathsf{P} = \mathsf{NP}$. We show that all Δ_2^{P}-hard sets are superterse unless $\mathsf{P} = \mathsf{NP}$.

Theorem 5.7.1 *Let B be an* NP*-complete set. The following four statements are equivalent:*

i. $\mathsf{P} = \mathsf{NP}$.

ii. *B is cheatable.*

iii. $(\exists k)[\mathrm{FQ}(k+1, B, \mathsf{P}) = \mathrm{FQ}(k, B, \mathsf{P})]$.

iv. $(\exists k)[\mathrm{FQ}_{||}(k+1, B, \mathsf{P}) = \mathrm{FQ}_{||}(k, B, \mathsf{P})]$.

Proof:

(i) ⇒ (ii,iii,iv): Assume that $\mathsf{P} = \mathsf{NP}$. Then $B \in \mathsf{P}$.

(ii) ⇒ (i): Assume that B is cheatable. Because B is NP-complete, SAT is m-reducible to B. Therefore, SAT is cheatable. Furthermore, SAT is self-tt-reducible because any Boolean formula can be reduced to the two Boolean formulas obtained by setting the first variable to 0 and to 1. By Theorem 5.4.6, SAT $\in \mathsf{P}$, so $\mathsf{P} = \mathsf{NP}$.

(iii) $\Rightarrow$ (ii): Follows from Theorem 5.4.3(i).

(iv) $\Rightarrow$ (ii): Follows from Theorem 5.4.3(ii).

∎

A set B is said to be NP-hard if all problems in NP are m-reducible to B.[1]

Corollary 5.7.2 *Let B be an NP-hard set. If $\mathsf{P} \neq \mathsf{NP}$ then*

i. B is not cheatable.

ii. $(\forall k)[\mathrm{FQ}(k, B, \mathsf{P}) \subset \mathrm{FQ}(k+1, B, \mathsf{P})]$.

iii. $(\forall k)[\mathrm{FQ}_{||}(k, B, \mathsf{P}) \subset \mathrm{FQ}_{||}(k+1, B, \mathsf{P})]$.

Proof:

i. By contradiction. Let B be an NP-hard set, and assume that B is cheatable. Let C be any NP-complete set. Then $C \leq_{\mathrm{m}} B$, so C is cheatable. By Theorem 5.7.1, $\mathsf{P} = \mathsf{NP}$, a contradiction.

ii. By contradiction. Assume that $\mathrm{FQ}(k+1, B, \mathsf{P}) = \mathrm{FQ}(k, B, \mathsf{P})]$. By Theorem 5.4.3(i), B is cheatable. This contradicts (i).

iii. By contradiction. Assume that $\mathrm{FQ}_{||}(k+1, B, \mathsf{P}) = \mathrm{FQ}_{||}(k, B, \mathsf{P})]$. By Theorem 5.4.3(ii), B is cheatable. This contradicts (i).

∎

Part (ii) was also proven by Krentel in [Kre88].

Let B be any NP-complete set. We have shown that extra queries to B allow us to compute extra functions in polynomial time, provided that $\mathsf{P} \neq \mathsf{NP}$. However, it is not known whether extra queries to B allow us to solve extra decision problems in polynomial time. For example, Blass and Gurevich [BG82], Valiant and Vazirani [VV86], and Papadimitriou and Yannakakis [PY84] have considered the

[1] Others have defined NP-hardness in terms of Turing reductions. The results to follow do not apply to that kind of NP-hardness.

class $\mathrm{D}^{\mathsf{P}} = \{L_1 - L_2 \mid L_1, L_2 \in \mathsf{NP}\}$; it is not known whether $\mathsf{P} \neq \mathsf{NP}$ implies that $\mathrm{D}^{\mathsf{P}} \neq \text{co-}\mathsf{NP} \cup \mathsf{NP}$.

It is well known that if $\mathsf{NP} = \text{co-}\mathsf{NP}$ then the Meyer-Stockmeyer polynomial time hierarchy [MS72, Sto77] collapses into NP. That observation relativizes.[2] Therefore, if we pick A to be an oracle such that $\mathsf{NP}^A = \text{co-}\mathsf{NP}^A$ but $\mathsf{P}^A \neq \mathsf{NP}^A$ [BGS75], then, computing relative to A, 1-query reducibility to an NP^A-complete set is identical with Turing reducibility to an NP^A-complete set. Thus, there is a relativized world in which $\mathsf{P} \neq \mathsf{NP}$, but extra queries to an NP-complete set do not allow us to solve extra decision problems. If B is NP-complete, Cai and Hemachandra [CH86] have constructed relativizations for each value of k that make $\mathsf{Q}_{||}(k, B, \mathsf{P}) \subset \mathsf{Q}_{||}(k+1, B, \mathsf{P}) = \mathsf{Q}_{||}(\omega, B, \mathsf{P})$.

The next result states that if extra queries to an NP-complete set do not allow us to solve extra decision problems in polynomial time, then all NP-hard sets are P-superterse, unless $\mathsf{P} = \mathsf{NP}$. We tend to disbelieve the hypothesis (because it implies that the polynomial-time hierarchy collapses [Kad88]) and we tend to believe the conclusion (because it is true under almost all relativizations [Bei87b]). However, neither belief has been proven true, and it is reassuring to know that at least one of them must be true, unless $\mathsf{P} = \mathsf{NP}$.

Theorem 5.7.3 *Assume that* $\mathsf{P} \neq \mathsf{NP}$*, and let* B *be a P-self-encoding set.*

i. If B *is* NP*-hard then* B *is P-superterse.*

ii. If B *is* NP*-complete then all* NP*-hard sets are P-superterse.*

Proof:

i. By contradiction. Assume that B is NP-hard and P-self-encoding, but not P-superterse. Since B is P-self-encoding but not P-superterse, B is cheatable, by Theorem 5.6.2. Since B is NP-hard and cheatable, $\mathsf{P} = \mathsf{NP}$, by Corollary 5.7.2. That is a contradiction.

[2] See [BGS75] by Baker, Gill, and Solovay for a discussion of relativizations.

ii. By (i) B is P-superterse. If C is NP-hard, then $B \leq_{\mathrm{m}} C$, so C must be superterse.

▮

We can relativize the bounded query classes for polynomial time in the same way as we relativized the bounded query classes in Chapter 4. We relativize some of the definitions from this Chapter.

Definition 5.7.4

- The set B is P^A-*superterse* if

$$(\forall C)(\forall n)[\mathrm{F}_n^B \notin \mathrm{FQ}^A(n-1, C, \mathsf{P})].$$

- The set B is P^A-*self-encoding* if

$$(\exists k)(\forall n)[\mathrm{Q}_{||}^A(n, B, \mathsf{P}) \subseteq \mathrm{Q}^A(k, B, \mathsf{P})].$$

- The set B is A-*cheatable* if

$$(\exists C)(\exists k)[\mathrm{F}_{2^k}^B \in \mathrm{FQ}^A(k, C, \mathsf{P})].$$

Theorem 5.7.5 *There is an oracle A such that all NP^A-hard sets are P^A-superterse.*

Proof: Proof by contradiction. There is an oracle A such that $\mathsf{NP}^A = \text{co-}\mathsf{NP}^A$ but $\mathsf{P}^A \neq \mathsf{NP}^A$ [BGS75]. Let C be any NP^A-complete set. Since $\mathsf{NP}^A = \text{co-}\mathsf{NP}^A$ the relativized polynomial time hierarchy collapses to NP^A, so

$$(\forall n)[\mathrm{Q}_{||}^A(n, C, \mathsf{P}) \subseteq \mathrm{Q}^A(1, C, \mathsf{P})].$$

Therefore, C is P^A-self-encoding. The proof of Theorem 5.7.3(ii) relativizes; since $\mathsf{P}^A \neq \mathsf{NP}^A$ and C is both NP^A-complete and P^A-self-encoding, all NP^A-hard sets are P^A-superterse. ▮

Thus we have found a relativization that makes all NP-hard sets P-superterse. In fact all NP-complete sets are P-self-encoding if and only if the Boolean Hierarchy of Wagner and Wechsung [WW85] collapses. Thus any oracle that collapses the Boolean Hierarchy [CH86] makes all NP-complete sets P-superterse. In [Bei87b], we have shown that all NP-hard sets are P-superterse under almost all relativizations.

Open Question 5.7.6

- *If* $\mathsf{P} \neq \mathsf{NP}$ *are all* NP*-hard problems P-superterse?*
- *Does there exist an oracle* A *such that* $\mathsf{P}^A \neq \mathsf{NP}^A$ *and some* NP^A*-hard problem is not* P^A*-superterse?*

An important class of problems is Δ_2^{P} (*i.e.*, P^{NP}, the class of all decision problems that can be solved in polynomial time with a polynomial number of queries to a SAT oracle). Two examples of sets that are complete for Δ_2^{P} are *Uniquely Optimal Traveling Salesperson* [Pap84] and *Odd Maximum Satisfying Assignment* [Kre88]. It is known that $\mathsf{P}^{\mathsf{P}^{\mathsf{NP}}} = \mathsf{P}^{\mathsf{NP}}$, so an unbounded number of queries to a Δ_2^{P}-complete oracle do not allow us to solve more decision problems in polynomial time than we can solve in polynomial time with just a single query to that oracle. Therefore all Δ_2^{P}-complete sets are P-self-encoding. By Theorem 5.7.3(i), every Δ_2^{P}-complete set is P-superterse unless $\mathsf{P} = \mathsf{NP}$; consequently every Δ_2^{P}-hard set is P-superterse unless $\mathsf{P} = \mathsf{NP}$. We formalize this proof below.

Theorem 5.7.7 *If* B *is* Δ_2^{P}*-hard, then* B *is P-superterse, unless* $\mathsf{P} = \mathsf{NP}$*.*

Proof: Let C be a Δ_2^{P}-complete set. We claim that C is P-self-encoding. Proof: Let $L \in \mathrm{Q}_{||}(n, C)$. Then L is decided by a polynomial time algorithm that makes n parallel queries to C. Since C is in $\Delta_2^{\mathsf{P}} = \mathsf{P}^{\mathsf{NP}}$, we can replace each query with an equivalent polynomial time computation that uses an NP-complete oracle. Thus L is decided by a polynomial time algorithm that uses an NP-complete oracle. Therefore $L \in \mathsf{P}^{\mathsf{NP}} = \Delta_2^{\mathsf{P}}$. Since C is Δ_2^{P}-hard, $L \leq_m C$; thus $L \in \mathrm{Q}(1, C)$. Therefore C is P-self-encoding.

By Theorem 5.7.3(i), C is P-superterse unless $\mathsf{P} = \mathsf{NP}$. Since B is Δ_2^{P}-hard, $C \leq_{\mathrm{m}} B$. Therefore B is P-superterse unless $\mathsf{P} = \mathsf{NP}$. ∎

5.8 Related Work

Amir and Gasarch [AG88] were the first to prove a bi-immunity result for 1-cheatable sets; they showed that if $A \subseteq 0^*$ is 1-cheatable then A or $0^* - A$ contains an infinite polynomial-time subset. In other words, no bi-immune tally set is 1-cheatable. In Theorem 5.4.10, we proved that result for sets over an arbitrary alphabet. This and similar problems are discussed in a survey paper [GJY87] by Goldsmith, Joseph, and Young.

A stronger version of Corollary 5.3.2 is proven in [ABG90]: If $\mathrm{F}^A_{2^k} \in \mathrm{FQ}(k, B, \mathsf{P})$ then every function that is Turing reducible to A in polynomial time is in $\mathrm{FQ}(k, B, \mathsf{P})$. We have generalized Theorem 5.4.6, by showing that if A is self-reducible and cheatable, then $A \in \mathsf{P}$. We have also shown that if A is cheatable then A is the union of a set in NP and a set that is polynomial time Turing reducible to a sparse oracle; consequently A is the union of a set in NP and a set that is accepted by a family of polynomial size circuits.

Wagner and Wechsung defined the Boolean Hierarchy

$$\mathsf{NP}(0), \text{co-}\mathsf{NP}(0), \mathsf{NP}(1), \text{co-}\mathsf{NP}(1), \ldots$$

in [WW85]. Cai and Hemachandra gave many equivalent definitions in [CH86]. We give another definition of the Boolean Hierarchy:

$$\begin{aligned} \mathsf{NP}(i) &= \{L \mid L \leq_{\mathrm{m}} \mathrm{PARITY}^{\mathrm{SAT}}_i\}, \\ \text{co-}\mathsf{NP}(i) &= \{\bar{L} \mid L \leq_{\mathrm{m}} \mathrm{PARITY}^{\mathrm{SAT}}_i\}. \end{aligned}$$

In [Bei91], we show that $L \in \mathrm{Q}_{||}(k, \mathrm{SAT})$ if and only if $L \in \mathrm{Q}(1, \mathrm{PARITY}^{\mathrm{SAT}}_k)$. Thus the bounded query classes relative to SAT are very closely related to the Boolean Hierarchy. Cai and Hemachandra [CH86] have constructed oracles that make the Boolean hierarchy collapse at arbitrary levels. They have also constructed oracles that make the hierarchy proper; Cai [Cai86] has shown that almost oracles make the hierarchy proper. Kadin [Kad88] has shown that if the Boolean hierarchy collapses then the polynomial time hierarchy collapses.

Book and Ko [BK88] have constructed, for each $k > 0$, a *sparse* set A such that for every sparse set B it is true that $\mathrm{Q}_{||}(k, A, \mathsf{P}) \subset \mathrm{Q}_{||}(k+1, B, \mathsf{P})$. This result does not hold if we remove either the sparseness condition or the time bound.

In [Kre88], Krentel considered polynomial time computations that are allowed to make $q(n)$ serial queries to an oracle, where n is the length of the input. He proved the following, stronger version of Corollary 5.7.2: If A is NP-complete and $q(n) \leq (1-\epsilon)\log n$ for some positive real number ϵ then

$$\mathrm{FQ}(q(n), A, \mathsf{P}) \subset \mathrm{FQ}(q(n)+1, A, \mathsf{P})$$

unless $\mathsf{P} = \mathsf{NP}$. He also showed that if A is NP-complete, $q(n) = O(\log n)$, and $\epsilon > 0$ then

$$\mathrm{FQ}(q(n), A, \mathsf{P}) \subset \mathrm{FQ}(n^{\epsilon}, A, \mathsf{P})$$

unless $\mathsf{P} = \mathsf{NP}$.

In [Bei87c], we consider polynomial time computations that are allowed to make $q(n)$ queries to an oracle, where n is the length of the input. We generalize Theorem 4.8.11, and we use that result to prove a generalization of Theorem 5.7.7, which states that if B is any Δ_2^{P}-complete oracle and

$$(\exists A)[\mathrm{FQ}_{||}(q(n), B, \mathsf{P}) \subseteq \mathrm{FQ}(q(n)-1, A, \mathsf{P})]$$

then $\mathrm{SAT} \in \mathrm{DTIME}(n^{O(q(n))})$.

Chapter 6

Conclusions

We have studied tradeoffs between serial queries to an oracle and parallel queries to an oracle. We have studied conditions under which $m > n$ queries to an oracle allow us to compute functions that we cannot compute by making only n queries to an oracle.

In Chapter 3, we showed that $2^n - 1$ parallel queries to K allow us to compute the same functions that we can compute by making n serial queries to K, where K is an oracle for the halting problem, *i.e.*,

$$\mathrm{FQ}_{||}(2^n - 1, K) = \mathrm{FQ}(n, K). \tag{10}$$

This result is not true for arbitrary oracles, because in [BGGO93] it was shown that there exists an oracle B such that $n+1$ parallel queries to B allow us to solve decision problems that we cannot solve by making only n serial queries to B, *i.e.*,

$$\mathrm{Q}_{||}(n+1, B) \not\subseteq \mathrm{Q}(n, B).$$

In addition, there exists an oracle B such that two serial queries to B allow us to solve more decision problems than we can solve by making only one round of parallel queries to B, *i.e.*,

$$\mathrm{Q}_{||}(\omega, B) \subset \mathrm{Q}(2, B).$$

There also exists an oracle A such that $n+1$ parallel queries to A allow us to compute functions that we cannot compute by making only n serial queries to any oracle B, *i.e.*,

$$(\forall B)[\mathrm{F}^{A}_{n+1} \notin \mathrm{FQ}(n, B)].$$

Thus equation (10) does not generalize in any way to arbitrary nonrecursive oracles. In [BGGO93], it is shown that equation (10) does not even apply to the jump of an arbitrary set, because for every nonrecursive set A

$$(\forall B)[\mathrm{F}^{A'}_{n+1} \notin \mathrm{FQ}(n, B)].$$

However, in the polynomial time bounded framework, equation (10) is half true, because

$$\mathrm{FQ}(n, B, \mathsf{P}) \subseteq \mathrm{FQ}_{||}(2^n - 1, B, \mathsf{P})$$

for every set B.

We have shown that $n + 1$ parallel queries to K allow us to solve more decision problems than we can solve by making only n parallel queries to K, *i.e.*,

$$\mathrm{Q}_{||}(n, K) \subset \mathrm{Q}_{||}(n + 1, K), \tag{11}$$

and that $n + 1$ serial queries to K allow us to solve more decision problems than we can solve by making only n serial queries to K, *i.e.*,

$$\mathrm{Q}(n, K) \subset \mathrm{Q}(n + 1, K). \tag{12}$$

In [BG87], we showed that these results are true for the jump of an arbitrary set, *i.e.*,

$$\mathrm{Q}_{||}(n, B') \subset \mathrm{Q}_{||}(n + 1, B') \text{ and } \mathrm{Q}(n, B') \subset \mathrm{Q}(n + 1, B');$$

however, neither equation (11) nor equation (12) generalizes to arbitrary nonrecursive oracles, because in [Bei87d] we constructed a nonrecursive set B such that one query to B allows us to solve every decision problem that we can solve by making n serial queries to B, *i.e.*,

$$\mathrm{Q}(n, B) = \mathrm{Q}(1, B)$$

for every n. We can generalize equations (11) and (12) as follows: For every nonrecursive oracle B and natural number n, $n + 1$ parallel queries to B allow us to compute more functions than we can compute by making only n parallel queries to B, *i.e.*,

$$\mathrm{FQ}_{||}(n, B) \subset \mathrm{FQ}_{||}(n+1, B),$$

and $n+1$ serial queries to B allow us to compute more functions than we can compute by making only n serial queries to B, *i.e.*,

$$\mathrm{FQ}(n, B) \subset \mathrm{FQ}(n+1, B).$$

The last two statements seem intuitively obvious; however, their proof depends on the Nonspeedup Theorem, which is not obvious.

We showed that n-weak-truth-table reducibility to K is equivalent to n-truth-table reducibility to K, *i.e.*,

$$B \leq_{n\text{-wtt}} K \Leftrightarrow B \leq_{n\text{-tt}} K.$$

We also classified the functions computable by making more than one round of parallel queries to K, showing that

$$\mathrm{FQ}_{||}(n_2, K) \circ \mathrm{FQ}_{||}(n_1, K) = \mathrm{FQ}_{||}((n_1+1)(n_2+1)-1, K).$$

In addition, we considered computations that are allowed to make an unbounded number of parallel queries during each round, thus obtaining a hierarchy of sets between those that are truth-table reducible to K and those that are Turing reducible to K.

In Chapter 4, we defined computability by a set of functions, and we showed that it captures the information-theoretic aspects of computability by a bounded number of queries to an oracle. This concept has been extremely useful in the study of bounded query classes. Using computability by a set of functions, we proved the Nonspeedup Theorem, which states that for every nonrecursive set A and every n it is not possible to answer 2^n parallel queries to A by making only n serial queries to another oracle B, *i.e.*,

$$(\forall B)[\mathrm{F}^A_{2^n} \notin \mathrm{FQ}(n, B)].$$

This is the tightest general result possible, by equation (10). In a sense, the Nonspeedup Theorem says that we cannot condense the information content of an oracle by more than a logarithmic amount.

If $\mathrm{F}^A_{2^n-1} \in \mathrm{FQ}(n, A)$ for all n, then we say that A is verbose. If $\mathrm{F}^A_{n+1} \notin \mathrm{FQ}(n, B)$ for any n and B, then we say that A is superterse. We showed that if a set A is not superterse then A is very far from being superterse, *i.e.*,

$$\mathrm{F}^A_{k+1} \in \mathrm{FQ}(k, B) \Rightarrow \mathrm{F}^A_n \in \mathrm{FQ}_{||}((k-2)\log n + O(1), C), \tag{13}$$

for some oracle C. In other words, this theorem says that if we can condense the information content of an oracle at all, then we can condense its information content by a logarithmic amount, within a constant.

In [BGGO93], it was shown that every truth-table degree contains a verbose set. Using that construction and the Nonspeedup Theorem, we showed that every truth-table degree contains a superterse set; we can also prove this result by using equation (13) and the Nonspeedup Theorem.

We proved the following surprising result: If k serial queries to the nonrecursive set B allow us to solve every decision problem that we can solve with n parallel queries to B for every n, then B is superterse, *i.e.*, if B is nonrecursive then

$$(\forall n)[\mathrm{Q}_{||}(n, B) \subseteq \mathrm{Q}(k, B)] \Rightarrow (\forall n)(\forall A)[\mathrm{F}^B_{n+1} \notin \mathrm{FQ}(n, A)].$$

The polynomial time bounded version of this result allowed us to show that all Δ_2^{P}-hard sets are P-superterse unless $\mathsf{P} = \mathsf{NP}$. It also allowed us to construct a relativization that makes all NP-hard sets P-superterse.

In Chapter 5, we described Amir and Gasarch's discovery [AG88] that the Nonspeedup Theorem is not valid for polynomial time bounded computations, because there exists a set $A \notin \mathsf{P}$ such that

$$(\forall n)[\mathrm{F}^A_n \in \mathrm{FQ}(1, A, \mathsf{P})].$$

We proved a Weak Nonspeedup Theorem, which states that if 2^k parallel queries to A can be answered by making only k queries to another oracle B, then any $n \geq 2^k$ parallel queries to A can be answered by making only $2^k - 1$ of the same queries in parallel, *i.e.*, if

$$(\exists B)[\mathrm{F}^A_{2^k} \in \mathrm{FQ}(k, B)],$$

then any $n \geq 2^k$ parallel queries to A can be answered by making only $2^k - 1$ of the same queries in parallel. If $\mathrm{F}^A_{2^k} \in \mathrm{FQ}(k, B)$ for some k and some B, then we say that A is cheatable.

Using the Weak Nonspeedup Theorem, we showed that every self-tt-reducible, cheatable set is in P. This allowed us to show that no NP-hard set is cheatable unless $\mathsf{P} = \mathsf{NP}$. We also showed that if B is self-tt-reducible but not in P or if B is NP-hard and $\mathsf{P} \neq \mathsf{NP}$, then $n+1$ parallel queries to B allow us to compute more functions than we can compute by making only n parallel queries to B, *i.e.*,

$$\mathrm{FQ}_{||}(n, B, \mathsf{P}) \subset \mathrm{FQ}_{||}(n+1, B, \mathsf{P}),$$

and $n+1$ serial queries to B allow us to compute more functions than we can compute by making only n serial queries to B, *i.e.*,

$$\mathrm{FQ}(n, B, \mathsf{P}) \subset \mathrm{FQ}(n+1, B, \mathsf{P}).$$

In [ABG90] we have shown that every self-reducible, cheatable set is in P.

Appendix A

Chromatic Number of a Recursive Graph

The theorem below was stated without proof in Section 3.7. We will prove it by constructing a prefix code for the natural numbers $0, \ldots, n$ and then applying Kraft's inequality [Gal68].

Theorem 3.7.5 *If there exists an oracle B and an algorithm that computes $\chi(G)$ for recursive graphs by making only $f(\chi(G))$ serial queries to B, then*

$$\sum_{i \geq 0} 2^{-f(i)} \leq 1.$$

Proof: Let $\mathcal{A}^B$ be an algorithm relative to B that computes $\chi(G)$ by making at most $f(\chi(G))$ serial queries to B, for some total recursive function f. Let

$$\chi_n(G) = \begin{cases} \chi(G) & \text{if } \chi(G) \leq n \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In [BG89], we showed that F_n^K is 1-query reducible to the function χ_n. By the Nonspeedup Lemma (4.2.3), F_n^K is not computable by a set of n partial recursive functions. Therefore, χ_n is not computable by a set of n partial recursive functions. We will use this fact later to obtain a contradiction.

For each sequence σ of k oracle answers, define a function $c_n^\sigma(G)$ computed as follows: Simulate $\mathcal{A}$ assuming that the sequence of oracle answers is σ; if $\mathcal{A}$ tries to make a $(k+1)$st query, if $\mathcal{A}$ outputs a number greater than n, or if $\mathcal{A}$ outputs a number i such that $k > f(i)$, then go into an infinite loop; otherwise, output the value output by $\mathcal{A}$. Since we can store the values $f(0), \ldots, f(n)$ in a finite table, c_n^σ is a partial recursive function for every n and σ. We write $\{0,1\}^*$ to denote the set of all sequences of bits. By the construction of c_n^σ, whenever $\chi_n(G)$ is defined

$$\chi_n(G) \in \{c_n^\sigma(G) \mid \sigma \in \{0,1\}^*\}. \tag{14}$$

Let σ be a prefix of σ'; because $\mathcal{A}$ is deterministic, if $c_n^\sigma(G)$ converges to a value then $c_n^{\sigma'}(G)$ must converge to the same value. We will use this fact later in order to construct a prefix code for the natural numbers 0 through n. By the construction of c_n^σ, if $c_n^\sigma(G)$ converges then

$$c_n^\sigma(G) \in \{0, \ldots, n\}.$$

Therefore,

$$(\forall n)(\forall G)[\{c_n^\sigma(G) \mid \sigma \in \{0,1\}^*\} \subseteq \{0, \ldots, n\}].$$

We claim that

$$(\forall n)(\exists G)[\{c_n^\sigma(G) \mid \sigma \in \{0,1\}^*\} = \{0, \ldots, n\}]. \tag{15}$$

Proof by contradiction. Suppose that

$$(\exists n)(\forall G)[\{c_n^\sigma(G) \mid \sigma \in \{0,1\}^*\} \subset \{0, \ldots, n\}].$$

Choose such a natural number n. Then

$$(\forall G)[\mathrm{card}(\{c_n^\sigma(G) \mid \sigma \in \{0,1\}^*\}) \leq n].$$

For $1 \leq j \leq n$, define a partial recursive function $h_j(G)$, computed as follows: Time-share $c_n^\sigma(G)$ for all σ until the functions have output j distinct values; output the jth distinct value. Therefore, for all G such that $\chi_n(G)$ is defined

$$\begin{aligned} \chi_n(G) &\in \{c_n^\sigma(G) \mid \sigma \in \{0,1\}^*\}) \qquad \text{by (14)} \\ &= \{h_j(G) \mid 1 \leq j \leq n\}. \end{aligned}$$

Thus the partial function χ_n is computable by a set of n partial recursive functions. This contradiction establishes the claim.

We write $|\sigma|$ to denote the length of the sequence σ. By (15), for every n, there exists a graph G such that for each i in $\{0, \ldots, n\}$, there exists a sequence σ_i of oracle answers such that $c_n^{\sigma_i}(G) = i$. By the definition of c_n^{σ}, it follows that $|\sigma_i| \leq f(i)$. As observed above, if $i \neq j$ then σ_i is not a prefix of σ_j. Therefore the sequences $\sigma_0, \ldots, \sigma_n$ form a prefix code for the natural numbers 0 through n. Therefore, by Kraft's Theorem [Gal68]

$$\sum_{0 \leq i \leq n} 2^{-|\sigma_i|} \leq 1.$$

Since $|\sigma_i| \leq f(i)$,

$$\sum_{0 \leq i \leq n} 2^{-f(i)} \leq 1.$$

Letting n approach infinity, we obtain the inequality

$$\sum_{i \geq 0} 2^{-f(i)} \leq 1.$$

∎

Bibliography

[ABG90] Amihood Amir, Richard Beigel, and William I. Gasarch. Some connections between bounded query classes and nonuniform complexity. In *Proceedings of the 5th Annual Conference on Structure in Complexity Theory*, pages 232–243, 1990.

[Add65] J. W. Addison. The method of alternating chains. In *Theory of Models*, pages 1–16, Amsterdam, 1965. North-Holland Publishing Co.

[AG88] Amihood Amir and William I. Gasarch. Polynomial terse sets. *Inf. & Comp.*, 77:37–56, April 1988.

[All86] Eric W. Allender. The complexity of sparse sets in P. In Alan L. Selman, editor, *Structure in Complexity Theory*, pages 1–11. Springer-Verlag, June 1986. Lecture Notes in Computer Science 223.

[Bei86] Richard Beigel. Query-limited reducibilities. Working draft, May 1986.

[Bei87a] Richard Beigel. Functionally supportive sets. Technical Report 87-10, The Johns Hopkins University, Dept. of Computer Science, 1987.

[Bei87b] Richard Beigel. SAT^A is terse with probability 1. Technical Report 87-04, The Johns Hopkins University, Dept. of Computer Science, 1987.

[Bei87c] Richard Beigel. A structural theorem that depends quantitatively on the complexity of SAT. In *Proceedings of the 2nd Annual Conference on Structure in Complexity Theory*, pages 28–32. IEEE Computer Society Press, June 1987.

[Bei87d] Richard Beigel. Supportive sets — II. Technical Report 87-15, The Johns Hopkins University, Dept. of Computer Science, 1987.

[Bei90] Richard Beigel. Unbounded searching algorithms. *SICOMP*, 19(3):522–537, June 1990.

[Bei91] Richard Beigel. Bounded queries to SAT and the Boolean hierarchy. *Theoretical Computer Science*, 84(2):199–223, July 1991.

[BG] Richard Beigel and William I. Gasarch. $O(\log n)$ verboseness. Manuscript in preparation.

[BG82] Andreas Blass and Yuri Gurevich. On the unique satisfiability problem. *Inf. & Control*, 55:80–88, 1982.

[BG87] Richard Beigel and William I. Gasarch. Supportive and parallel-supportive sets. Technical Report 1805, University of Maryland, Dept. of Computer Science, 1987.

[BG89] Richard Beigel and William I. Gasarch. On the complexity of finding the chromatic number of a recursive graph I: The bounded case. *Annals of Pure and Applied Logic*, 45(1):1–38, November 1989.

[BGGO93] Richard Beigel, William I. Gasarch, John T. Gill, and James C. Owings. Terse, superterse, and verbose sets. *Inf. & Comp.*, 103:68–85, 1993.

[BGH89] Richard Beigel, William I. Gasarch, and Louise Hay. Bounded query classes and the difference hierarchy. *Archive for Mathematical Logic*, 29(2):69–84, December 1989.

[BGO89] Richard Beigel, William I. Gasarch, and James C. Owings, Jr. Nondeterministic bounded query reducibilities. *Annals of Pure and Applied Logic*, 41(2):107–118, 1989.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the $P =?\ NP$ question. *SICOMP*, 4:431–442, 1975.

[BH77] L. Berman and J. Hartmanis. On isomorphism and density of NP and other complete sets. *SICOMP*, 6:305–322, 1977.

[BK88] Ronald V. Book and Ker-I Ko. On sets truth-table reducible to sparse sets. *SICOMP*, 17:903–919, 1988.

[BS85] J. L. Balcázar and U. Schöning. Bi-immune sets for complexity classes. *MST*, 18:1–10, 1985.

[BY76] Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *IPL*, 5(3):82–87, August 1976.

[Cai86] Jin-yi Cai. *On Some Most Probable Separations of Complexity Classes.* PhD thesis, Cornell University, Ithaca, NY, 1986.

[Car62] Lewis Carroll. *Alice's Adventures in Wonderland and Through the Looking Glass.* Macmillan Publishing Co., Inc., New York, 1962.

[CH86] Jin-yi Cai and Lane A. Hemachandra. The Boolean hierarchy: Hardware over NP. In Alan L. Selman, editor, *Structure in Complexity Theory*, pages 105–124. Springer-Verlag, June 1986. Lecture Notes in Computer Science 223.

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

[EHK81] Richard L. Epstein, Richard Haas, and Richard L. Kramer. Hierarchies of sets and degrees below $\mathbf{0}'$. In *Logic Year 1979–80*, volume 859 of *Lecture Notes in Mathematics*, pages 32–48, Berlin, 1981. Springer-Verlag. Volume 859 of *Lecture Notes in Mathematics.*

[Eps79] Richard L. Epstein. *Degrees of Unsolvability: Structure and Theory*, volume 759 of *Lecture Notes in Mathematics.* Springer-Verlag, Berlin, 1979.

[Ers68a] Yu. L. Ershov. A hierarchy of sets, I. *Algebra i Logika*, 7(1):47–74, January–February 1968. English Translation, Consultants Bureau, NY, pp. 25–43.

[Ers68b] Yu. L. Ershov. A hierarchy of sets, II. *Algebra i Logika*, 7(4):15–47, July–August 1968. English Translation, Consultants Bureau, NY, pp. 212–232.

[Ers70] Yu. L. Ershov. A hierarchy of sets, III. *Algebra i Logika*, 9(1):34–51, January–February 1970. English Translation, Consultants Bureau, NY, pp. 20–31.

[Gal68] P. E. Gallager. *Information Theory and Reliable Communication.* Wiley, New York, 1968.

[Gas86] William I. Gasarch, 1986. Personal communication.

[Gas87] William I. Gasarch, 1987. Personal communication.

[GJY87] Judy Goldsmith, Deborah Joseph, and Paul Young. Self-reducible, p-selective, near-testable, and p-cheatable sets: The effect of internal structure on the complexity of a set. Technical Report 87-06-02, Dept. of Computer Science, University of Washington, Seattle, June 1987. An extended abstract appeared in *Proceedings of the 2nd Annual Conference on Structure in Complexity Theory*, IEEE Computer Society Press, June 1987, pp. 50–59.

[Gol65] E Mark Gold. Limiting recursion. *JSL*, 30(1):28–48, March 1965.

[Hay78] Louise Hay. Convex subsets of 2^n and bounded truth-table reducibility. *Discrete Mathematics*, 21(1):31–46, January 1978.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[HY84] J. Hartmanis and Y. Yesha. Computation times of NP sets of different densities. *Theoretical Computer Science*, 34:17–32, 1984.

[Joc68] C. G. Jockusch, Jr. Semirecursive sets and positive reducibility. *T. AMS*, 131:420–436, May 1968.

[Kad88] Jim Kadin. The polynomial time hierarchy collapses if the Boolean hierarchy collapses. *SICOMP*, 17(6):1263–1282, December 1988.

[Knu81] Donald E. Knuth. Supernatural numbers. In David A. Klarner, editor, *The Mathematical Gardner*, pages 310–325. Wadsworth International, Belmont, California, 1981.

[Kre88] Mark W. Krentel. The complexity of optimization problems. *JCSS*, 36(3):490–509, 1988.

[Lac65] Alistair H. Lachlan. Some notions of reducibility and productiveness. *Zeitsch. f. math. Logik und Grundlagen d. Math.*, 11:17–44, 1965.

[LMF76] N. A. Lynch, A. R. Meyer, and M. J. Fischer. Relativization of the theory of computational complexity. *T. AMS*, 220:243–287, 1976.

[MS72] A. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th Annual IEEE Symposium on Switching and Automata Theory*, pages 125–129, 1972.

[MY78] Michael Machtey and Paul Young. *An Introduction to the General Theory of Algorithms*. Elsevier North-Holland, Inc., New York, 1978.

[Odi81] Piergiorgio Odifreddi. Strong reducibilities. *Bulletin of the American Mathematical Society*, 4(1):37–86, 1981.

[Owi86] James C. Owings, Jr., 1986. Personal communication.

[Owi89] James C. Owings, Jr. A cardinality version of Beigel's Nonspeedup Theorem. *JSL*, 54(3):761–767, September 1989.

[Pap84] Christos H. Papadimitriou. On the complexity of unique solutions. *J. ACM*, 31(2):392–400, April 1984. Also appeared in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 14–20, 1982.

[Put65] Hilary Putnam. Trial and error predicates and the solution to a problem of Mostowski. *JSL*, 30(1):49–57, March 1965.

[PY84] C. H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *JCSS*, 28:244–259, 1984.

[Rab60] M. O. Rabin. Degree of difficulty of computing a function and a partial ordering of recursive sets. Technical Report 2, The Hebrew University, Jerusalem, 1960.

[Rog67] Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw Hill, New York, 1967.

[Sac61] G. E. Sacks. A minimal degree less than $\mathbf{0}'$. *Bulletin of the American Mathematical Society*, 67:416–419, 1961.

[Sch76] C. P. Schnorr. Optimal algorithms for self-reducible problems. In *Proceedings of the 3rd International Colloquium on Automata, Languages, and Programming*, pages 322–337, 1976.

[Soa87] Robert I. Soare. *Recursively Enumerable Sets and Degrees.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1987.

[Sto77] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[VV86] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[WW85] G. Wechsung and K. Wagner. On the Boolean closure of NP. In *Proceedings of the 1985 International Conference on Fundamentals of Computation Theory*, pages 485–493. Springer-Verlag, 1985. Volume 199 of *Lecture Notes in Computer Science*.