INFORMATION TO USERS

While the most advanced technology has been used to photograph and reproduce this manuscript, the quality of the reproduction is heavily dependent upon the quality of the material submitted. For example:

- Manuscript pages may have indistinct print. In such cases, the best available copy has been filmed.
- Manuscripts may not always be complete. In such cases, a note will indicate that it is not possible to obtain missing pages.
- Copyrighted material may have been removed from the manuscript. In such cases, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, and charts) are photographed by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is also filmed as one exposure and is available, for an additional charge, as a standard 35mm slide or as a 17"x 23" black and white photographic print.

Most photographs reproduce acceptably on positive microfilm or microfiche but lack the clarity on xerographic copies made from the microfilm. For an additional charge, 35mm slides of 6"x 9" black and white photographic prints are available for any photographs or illustrations that cannot be reproduced satisfactorily by xerography.

Order Number 8721886

On the structure of intractable sets

Geske, John George, Ph.D.

Iowa State University, 1987

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

On the structure of intractable sets

by

John George Geske

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University

Ames Iowa

1987

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Computational complexity deals with the fundamental issues of determining the intrinsic difficulty of mathematically posed problems. Typically, the complexity of a problem is measured by the amount of computational resources — usually time or memory — needed to solve the problem. This presupposes that we have a precise notion of a "computable problem" and a realistic model of computation on which to measure computational resources; that we have both is due to Alan Turing [66].

Turing invented a mathematically simple device — the ubiquitous Turing machine — and proceeded to show that there existed a well-defined mathematical set whose membership problem could not be determined by this machine. He argued that this device precisely captured the notion of "computable" and that no process, either human or mechanical, could therefore determine set membership for this particular set; this problem was *undecidable.* While his arguments generated considerable controversy, every attempt to precisely define the notion of computable has been shown to be equivalent to Turing's definition. This invariance of the Turing machine is formulated in *Church's Thesis*:

> *Every procedure is effective (an algorithm) if and only if it can be simulated (computed) on a Turing machine.*

Belief in this thesis is a matter of faith, but as pointed out by Rogers [57] and Webb [69] a large body of empirical evidence suggests that our faith is not misplaced. We are confident, therefore, in using "Turing computable" as a precise definition of computable and the Turing machine as our model of computation.

It soon became apparent that the class of decidable problems did not accurately define those problems which were feasible to compute in any realistic time-frame, [1] but

[1] Von Neumann [68] was apparently aware of this as early as 1953.

it was not until the early 1960s that a precise definition of feasible computations was made. Cobham [16] considered a set to be tractable only if there existed an algorithm for accepting it that ran in polynomial time. Similarly, Edmonds [20] equated the polynomial time algorithms with "good" algorithms.

The choice of polynomial time might seem somewhat arbitrary; a program with run time bound of n^{100}, for example, hardly seems tractable. Nevertheless, there is wide agreement that a problem has not been well-solved unless there exists a polynomial time-bounded algorithm for doing it. Polynomial time is also appealing since the class of polynomials is closed under addition, multiplication and composition. We may concatenate programs that run in polynomial time, iterate them a polynomial number of times, or allow them to make subroutine calls to another polynomial time-bounded program, and the program still runs in polynomial time.

Our faith in using polynomial time-bounded Turing machines as a model of computational complexity rests on a variant of Church's Thesis:

> *An algorithm is executable in polynomial time if and only if there is an equivalent Turing machine that operates in polynomial time.*

If Church's Thesis is the foundation of computer science, then this polynomial time variant must surely be considered the keystone to the study of computational complexity. While this variant, like Church's Thesis, is unprovable, there is strong evidence that it is true; known simulations of random access machine models by Turing machines, for example those of Aho, Hopcroft and Ullman [2] or Machtey and Young [48], cause computations to be slowed up by at most a polynomial factor.

The complexity class P is the class of all computable sets that are recognized by polynomial time-bounded Turing machines. A problem is said to be intractable if it is so hard that *no* polynomial time algorithm can possibly solve it. Obviously the undecidable sets constructed by Turing are intractable; the earliest examples of intractable decidable sets were obtained by Hartmanis and Stearns [36]. Meyer and Stockmeyer [50], Ferrante and Rackoff [24], and others first proved the existence of "natural" decidable problems that were intractable. These problems are provably not in NP, the class of all sets recognized by polynomial time-bounded nondeterministic

Turing machines. Inasmuch as P $\neq$ NP is an open problem, all the known provably intractable problems are either undecidable or out of NP.

The complexity class NP was first studied by Cook [18]. He introduced a polynomial time-bounded restriction of Turing reducibility in order to show that every set in NP is reducible in this restricted sense to the Satisfiability Problem. Karp [41] strengthened and extended the work of Cook. He defined a polynomial time-bounded restriction of many-one reducibility, and showed that every set in NP could be reduced to any of a number of classic combinatorial problems by this reducibility. These problems themselves belonged to NP. There was a brief period of unsettlement over notation, but of course, such problems are now called NP-complete, and the reducibilities formulated by Cook and Karp are called *efficient reducibilities.* Therefore, to obtain faster (tractable) algorithms for these problems is to prove that P = NP.

Few problems in Computer Science have generated as much interest and debate as has the P $\neq$ NP problem. This problem has come to dominate the field of computational complexity. It has driven an enormous number of researchers in the study of combinatorial algorithms, and new and important fields of research such as approximation algorithms and probabilistic machines can trace their origins to the study of this central problem. Nevertheless, the P $\neq$ NP problem remains one of the outstanding open problems in Computer Science.

We do not propose to tackle this problem here. Rather, we are interested in studying the structural relations between intractable sets and the complexity class NP. The first systematic study of efficient reducibilities is due to Ladner, Lynch, and Selman [45]. They introduced the notation $\leq_m^P$, and $\leq_T^P$ for polynomial time-bounded many-one reducibility and polynomial time-bounded Turing reducibility, respectively, and they proved, for example, that $\leq_m^P$ and $\leq_T^P$ are distinct on sets in DTIME(2^n). With some exception, the work by Selman on p-selective sets [63], [64] being notable here, it is not known whether $\leq_m^P$ and $\leq_T^P$ differ on NP. It is not known whether NP-complete sets are p-isomorphic. *Absolute* structural results for NP appear as difficult to arrive at as resolution of the P $\neq$ NP problem — a point we will expand upon shortly — and so we examine structural properties of other natural complexity classes, E, the linear exponential time-bounded class, and TIME(2^{poly}), in hopes that these results will shed light on the structural aspects of NP.

We extend the traditional approach of using efficient reducibilities to study structural relationships between computable sets. We define a nonconstructive binary relation that makes precise the notion that "the complexity of A is polynomially related to the complexity of B." The approach taken here to define the polynomial complexity of A is analogous to the approach taken by mathematicians in defining the cardinality of a set A. This is the weakest mathematically meaningful notion that captures all other polynomial reducibilities, including polynomial time Turing reducibility.

Here we are motivated by nothing less than a reexamination of what it means for a set to be NP-complete. Are there sets in NP that in a mathematically meaningful sense should be considered to be complete for NP, but that are not NP-complete in the usual sense that every set in NP is $\leq_m^P$-reducible to it? While we fall short of answering our question about completeness for NP, we do show that there are sets that are hard for NP that are not NP-hard in the usual sense, and we do show that there are sets that must be considered to be complete for the complexity class E that are not even $\leq_T^P$-complete for E.

In a certain way, hardness and completeness with respect to this relation is related to the notion of an *almost everywhere (a.e.) complex* set, and so we initiate this investigation by first studying a.e. complex sets. Recall that if a set $A \notin \mathrm{DTIME}(t(n))$ for some time function t, then for every deterministic Turing machine M that recognizes A there exist infinitely many inputs x for which the running time of M on x is slower than $t(|x|)$. This leaves open the possibility that there exists some other infinite set of inputs which M can recognize within the time bound t; the set A is *infinitely often (i.o.) complex with respect to* t. The set $A \notin \mathrm{DTIME}(t(n))$ a.e if for every deterministic Turing machine M that recognizes A, M runs for more than $t(|x|)$ steps for all but finitely many strings x. Therefore, the a.e. complex sets are intractable in the severest sense. Significantly, we show that a.e. complex sets can be constructed almost as easily as i.o. complex sets.

Ko, Orponen, Schöning, and Watanabe [43] pointed out that there are two principal views on the cause of intractability. The "distributional" view suggests that the hard instances of an intractable problem are distributed in some irregular manner, but feasible algorithms can only determine "smooth" distributions. The other view, as discussed by Hartmanis [34], is that it is the individual instances of an intractable

problem that are inherently hard, *i.e.*, hard independent of any particular algorithm to decide the problem. The work done here strengthens this view of *instance complexity*. We find, by analysis of our newly defined notion of completeness and hardness, that the common existence of a.e. complex sets in E and TIME(2^{poly}) places restrictions on the distribution of hard instances for the $\leq_m^P$-complete sets for these classes.

We research into these theoretical topics in the belief that there is a deep mathematical structure that needs to be understood before the hard questions of wider interest can be resolved. This belief is based, in part, on a historical analogy. Our *modus operandi* is based on the failure of this analogy.

It is commonly believed that the question of whether P $\neq$ NP has certain structural similarities to Post's problem — the problem in pure recursion theory of whether there are incomplete r.e. sets. As Young [70] has eloquently stated:

> It has been argued that the problem P *vs.* NP bears a reasonable analogy to Post's problem and that, just as Post's problem required the invention of essentially new techniques for its solution, so too will P *vs.* NP require essentially new techniques for *its* solution. It does indeed appear that new techniques will be required if P *vs.* NP is to be solved, but it should be noted that Post's problem was not solved by looking at "natural" problems, but was instead solved by using a structural approach. Post's problem was definitively stated in 1944, and its solution took a remarkably short time. Within little more than a dozen years Friedberg and Muchnik had invented the necessary techniques. In the intervening years, structural problems for r.e. sets were intensely investigated and a remarkably rich and beautiful theory developed regarding the structure of r.e. sets and the reducibilities among these sets. Although the Friedberg and Muchnik solution perhaps explicity used very little of this theory, it is difficult to believe that their structural solutions could have arisen in a vacuum: Without the preceding intense research on the structural properties of r.e. sets it is difficult to imagine the scientific, social and historical context which would have permitted the Friedberg-Muchnik solutions.

Indeed, it is very seductive to consider polynomial time complexity to be analogous to classical recursion theory; the classes P and NP being analogous to the classes of recursive and recusively enumerable sets, respectively; the NP-complete sets the direct analogue of the complete r.e. sets; the polynomial time hierarchy the direct analogue of the arithmetical hierarchy.

Ladner [44] showed that the structure of $\leq_T^P$-degrees in NP is similar to the structure of recursively enumerable $\leq_T$-degrees. For example, they both form a dense upper semi-lattice. The approach taken by Ladner, Lynch and Selman [45] followed classical lines; an attempt was made to distinguish polynomial time-bounded reducibilities modeled after the development of effective reducibilities in recursive function theory due to Post [54]. The p-selective sets used by Selman [63], [64] is based on the McLaughlin and Martin construction that appears in Jockusch [40]. Some properties of this construction have exact analogues in NP while others fail altogether. Even the Hartmanis-Berman conjecture [10] — the conjecture that all NP-complete sets are p-isomorphic — is based on the fact that in recursive function theory all complete sets are recusively isomorphic.

But these analogies are not quite correct. Breidbart [15], for instance, has shown that no infinite, coinfinite maximal language can exist in NP. In this respect, the nondeterministic complexity classes resemble the deterministic complexity classes (and the recursive sets) more than they resemble the recusively enumerable sets. Also, it is well known that the analogue of Post's theorem fails; there exist recursive sets A and B such that A and $\overline{A}$ are both NP in B, but A is not deterministically reducible to B in polynomial time.

Even the techniques of recursion theory seem to fail. As seen, early researchers optimistically set out to solve the $\mathrm{P} \neq \mathrm{NP}$ problem employing these techniques. It seemed reasonable to assume that any diagonalization technique yielding $\mathrm{P} \neq \mathrm{NP}$ would be sufficiently general to yield, for every set X, $\mathrm{P}^X \neq \mathrm{NP}^X$, and that any simulation technique yielding $\mathrm{P} = \mathrm{NP}$ would be sufficiently general to yield for every set X, $\mathrm{P}^X = \mathrm{NP}^X$. Baker, Gill and Solovay [6] burst this balloon of optimism by showing the existence of sets A and B such that $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$, so that $\mathrm{P} = \mathrm{NP}$ does not imply that for every set X, $\mathrm{P}^X = \mathrm{NP}^X$, and $\mathrm{P} \neq \mathrm{NP}$ does not imply that for every set X, $\mathrm{P}^X \neq \mathrm{NP}^X$.

The resultant flood of *relativization results* that followed this seminal paper cast a pall over the entire complexity field. Answers about complexity class containment for all but the most trivial of cases seem to be out of reach; the structural results alluded to by Young have been shown, by Homer and Maass [38], to be difficult to achieve. These results are often cited as proof that new techniques must be developed

before absolute results can be found. Unfortunately, "in spite of the fact that recursive function theory has been a rich source of ideas, concepts, and questions, it has been a notorious failure in providing solutions about computational problems in NP" [70].

And so we have the following dichotomy: On the one hand we are tantalized by the methods and results of recursive function theory; on the other hand we are faced with the failure of these techniques to solve the analogous problems in complexity theory. This dissertation is similarly dichotomous. While we whole-heartedly embrace the belief that the study of structural characteristics of intractable sets is necessary to gain the insight needed to solve the more global questions, we reject out of necessity, as the relativization results have shown, the techniques and mechanisms of recursive function theory.

The failure of traditional techniques is the *raison d'être* of the second part of this dissertation. Here we provide a few more drops to the ocean of relativization results. Information on classes relativized to oracles can often lend plausability to conjectures about the nonrelativized classes — conjectures which currently defy solution. In particular, we establish several translation and separation results for relativized subclasses of NP.

We first look at whether the recent result of Paul, Pippenger, Szemerédi and Trotter [53] that deterministic real-time differs from nondeterministic real-time can be of any help in proving P $\neq$ NP. Book [11] observed that P = NP if and only if NTIME(n) $\subseteq$ P. Therefore, the result of Paul, *et al.*, is a special case of the, as yet unproven, assertion that P $\neq$ NP. We show that for every i there exists a recursive oracle A such that $\text{NTIME}(n)^A \not\subseteq \text{DTIME}(O(n^i))^A$ and $\text{NTIME}(n)^A \subseteq \text{DTIME}(O(n^{i+1}))^A$. As a corollary, we prove the existence of a recursive oracle A such that $\text{DTIME}(O(n))^A \neq \text{NTIME}(n)^A$ and $\text{P}^A = \text{NP}^A$.

According to the traditional point of view about relativization results, known proof techniques will not succeed in obtaining separation results when there are oracles for which the relativized classes are the same and oracles for which the relativized classes differ, although this view needs to be tempered due to the recent results of controlled relativization by Book, Long and Selman [13]. Though no precise rendering has been made, recursion theoretic techniques usually relativize, while combinatorial techniques do not necessarily relativize, and it is worth noting, in this context, that

the results of Paul, *et al.* are heavily combinatoric.

Next we study relationships between P, NP, and the unambiguous and random time classes UP and RP. Questions concerning these relationships are motivated by complexity issues in public-key cryptography. We prove that there exists a recursive oracle A such that $P^A \neq UP^A \neq NP^A$, and such that the first inequality is "strong," i.e., there exists a P^A-immune set in UP^A. Further, we construct a recursive oracle B such that UP^B contains an RP^B-immune set. As a corollary, we obtain $P^B \neq RP^B \neq NP^B$ and both inequalities are strong. By use of the techniques employed in the proof that $P^A \neq UP^A \neq NP^A$, we are also able to solve an open problem raised by Book, Long and Selman [13].

The mathematical notation and definitions used throughout this work is presented in Chapter 2. We introduce the Turing machine as our model of computation and discuss the reasons why this device is an appropriate vehicle to study computational complexity.

In Chapter 3, we state and prove a deterministic time hierarchy theorem for a.e. complexity that is as tight as the well known Hartmanis-Stearns hierarchy theorem for i.o. complexity. This result is a significant improvement over all previously known hierarchy theorems for a.e. complex sets. As a corollary we obtain a simplified proof of a similar hierarchy theorem for determinisitic space that was first proven by Meyer and McCreight [49]. These results are due to a collaboration with D. Huynh (Geske and Huynh [30]).

We derive similar, very tight, hierarchy theorems for sets that cannot be a.e. complex for syntactic reasons, but for which, intuitively, a.e. complex notions should exist. For example, no subset of $\{1\}^*$ can be a.e. complex, but we show, essentially, that for every two deterministic time classes C_1 and C_2, if C_1 contains a language that is not in C_2, then C_1 contains a tally language L such that no infinite subset of L belongs to C_2 and no infinite tally language in the complement of L is in C_2. Similar results are applied to the study of P-printable sets and to sets of low generalized Kolmogorov complexity (See, for example, Allender [3], Allender and Rubinstein [4], and Hartmanis [33]).

We define, in Chapter 4, a nonconstructive binary relation that relates, in a natural way, the computational complexity of two recursive sets. This relation is

the weakest notion that captures all other polynomial time reducibilities and is even weaker than the reducibility defined by Even, Long and Yacobi [21]. The equivalence classes of this relation are called *polynomial complexity degrees.*

We show that this relation is properly weaker than polynomial time Turing reducibility and yields new completeness and hardness notions for complexity classes. These results provide new insights into the structure of the "hard instances" of complete sets, and we show that the hard instances of complete sets for E and TIME(2^{poly}) have fairly regular distribution.

We begin our investigation of relativization in Chapter 5. Does the fact that DTIME($O(n)$) $\neq$ NTIME(n) help in leading us to a proof that P $\neq$ NP? Does one imply the other? We seek evidence that this is hard. We construct an oracle that answers this question in the affirmative, and we construct an oracle that answers this question in the negative. We conclude from from our relativization results that the result of Paul *et al.* does not imply P $\neq$ NP by recursive theoretic techniques.

Finally, in Chapter 6, we investigate the containment relationships between the common complexity classes P and NP with the more specilized classes UP and RP. Even, Selman, and Yacobi [22] showed that the question of whether UP was equivalent to NP is closely related to the question of whether there exist NP-hard public-key cryptosystems. Grollmann and Selman [32] showed that "secure" cryptosystems should have RP-immune sets in UP. We construct oracles that separate UP and RP from P and NP and from each other, and most of these separations are strong. These results involve a combinatorial argument for which we have developed a pebbling game. A natural generalization of this game is used in solving an open problem of Book, Long, and Selman [13]. The results presented here are due, in part, to a collaboration with J. Grollmann (Geske and Grollmann [29]).

We summarize the main points of our work in Chapter 7, list open problems that have been raised, and present some remarks about directions for future work.

2 PRELIMINARIES

2.1 Conventions and Notation

Let $\mathbf{N}$ denote the set of nonnegative integers, Σ the binary alphabet $\{0,1\}$ and Σ^* the set of strings over Σ. *Languages* are subsets of Σ^*. $|w|$ denotes the length of the string $w \in \Sigma^*$. We assume a total ordering on $\Sigma^* = \{w_i \mid i \geq 1\}$ such that shorter strings precede longer ones, and strings of the same length are ordered lexicographically. We will adopt the notation $\Sigma^{\leq n}$ (respectively, Σ^n) for the set of all strings in Σ^* of length at most n (respectively, exactly n). For any set $A \subseteq \Sigma^*$, the *complement* of A, $\Sigma^* - A$, is denoted by $\overline{A}$. If $\mathcal{C}$ is a class of sets, co-$\mathcal{C}$ denotes the class $\{\overline{A} | A \in \mathcal{C}\}$. The cardinality of the set A is denoted by $\|A\|$.

A set $A \subseteq \Sigma^*$ is *sparse* if there is a polynomial p such that for all positive integers n, $\|\Sigma^n \cap A\| \leq p(n)$. A set A is a *tally set* if $A \subseteq \{1\}^*$.

A *lattice* is a partially ordered set in which every pair of elements has both a least upper bound and a greatest lower bound. An *upper semi-lattice* is a partially ordered set in which every two elements have a least upper bound.

Let f be a function, $f : \Sigma^* \to \Sigma^*$. The function f is *finite-one* if for every $y \in \Sigma^*$, $f^{-1}(y)$ is finite. The function f is *honest* if there exists a polynomial p such that for all $x \in \Sigma^*$, $p(|f(x)|) \geq |x|$.

We write $\exists^\infty x$ to denote "for infinitely many x," and $\forall^\infty x$ to denote "for all except possibly finitely many x."

We use the notation $O(f(n))$ to stand for any function whose magnitude is upper-bounded by a constant times $f(n)$, for all large n,*e.g.*, $O(f(n))$ denotes the set of all functions g such that there exist positive constants c and n_0 with $g(n) \leq cf(n)$ for all $n > n_0$.

In an inductive construction of a set X, $X(i)$ denotes the finite set of strings

placed into X prior to stage i.

For $m, n \in \mathbf{N}$ we define $\langle m, n \rangle = \frac{1}{2}[(m+n)(m+n+1)] + m$ and inductively $\langle m_1, \ldots, m_{k+1} \rangle = \langle \langle m_1, \ldots, m_k \rangle, m_{k+1} \rangle$. Given $m_1 \ldots, m_k$, $\langle m_1, \ldots, m_k \rangle$ can be determined in polynomial time as a function of the sum of the lengths of $m_1, \ldots, m_k$ written in binary. For a fixed k there is a polynomial time-bounded algorithm (as a function of the length of m written in binary) for determining $m_1, \ldots, m_k$ such that $m = \langle m_1, \ldots, m_k \rangle$.

2.2 Computability and Complexity

We use the Turing machine as our model of computation. Although a simple device, a Turing machine is as powerful as any other computing machine. We do not formalize Turing machines here (See for example Hopcroft and Ullman [39]). Typically, when we need to demonstrate the existence of a specific machine we present an algorithm written in a pseudo-Pascal language, confident that a Turing machine exists that implements the algorithm. Based on our faith in Church's Thesis we will use the terms "algorithm", "procedure" and "machine" interchangably. When the particular model of Turing machine is relevent we assume a *multi-tape offline Turing transducer*, *i.e.*, a Turing machine with a fixed number of work tapes, a single read-only input tape, and a single write-only output tape. A Turing machine may be *deterministic* or *nondeterministic*. Informally, a Turing machine is nondeterministic if after each step in a computation it is allowed to have more than one possible next step.

A Turing machine M *accepts* a string x if there exists a computation of M on input x that halts in a distinguished accepting state. $L(M)$ denotes the set of strings accepted by transducer, and M is called an *acceptor* of the set $L(M)$. A Turing machine acceptor M is a *recognizer* if M has both accepting and rejecting states such that on every input x there exists a computation of M that halts in either an accepting or rejecting state. A transducer M *computes* a value y on an input string x if there is an accepting computation of M on x for which y is the current contents of M's output tape. We will assume a standard, fixed encoding of Turing machines into strings. The machine encoded by the natural number i is denoted M_i; φ_i denotes

the function computed by transducer M_i. The sequence $M_1, M_2, \ldots$ is an *effective enumeration* of Turing machines.

Let $t : \mathbf{N} \to \mathbf{N}$ be a function. A Turing machine M is $t(n)$ *time(space)-bounded* if any computation of M on input of length n performs (visits) at most $t(n)$ steps (tape cells on any worktape). The function t is *fully time(space)-constructible* if there is some Turing machine M that is $t(n)$ time(space)-bounded, and M performs (uses) exactly $t(n)$ steps (cells) on all inputs of length n. A function $f : \Sigma^* \to \Sigma^*$ is said to be computable in time (space) $t(n)$ if there is a $t(n)$ time(space)-bounded transducer M that on input x outputs $f(x)$ in its accepting state.

We let $T_M(x)$ denote the running time of machine M on input x. We adopt the convention that "time complexity $t(n)$" means $\max(n+1, \lceil t(n) \rceil)$ and "space complexity $t(n)$" means $\max(1, \lceil t(n) \rceil)$.

Given a fully time-constructible function t, any Turing machine M can be converted to a $t(n)$ time-bounded Turing machine in the following fashion: Let M_1 be a Turing machine that executes exactly $t(n)$ steps on all inputs of length n and then halts; M_1 exists by virtue of t being fully time-constructible. Let M_2 be a Turing machine with two independent read heads on the input tape. On input x, M_2 simulates in parallel successive steps of M and M_1 on input x. This simulation continues until either M accepts or rejects x, whereupon M_2 mimics M, or M_1 halts after exactly $t(|x|)$ steps, at which time M_2 rejects x and halts. Using the techniques of Fischer, Meyer and Rosenberg [25], we can convert M_2 to an equivalent Turing machine with a single read head on the input tape that runs at precisely the same speed as M_2. Note that M_1 acts as a "clock," terminating every computation of M that exceeds $t(n)$ steps on inputs of length n. Therefore, for every fully time-constructible function t, there exists an effective enumeration of languages in DTIME(f) and NTIME(f):

$$\begin{aligned}
\text{DTIME}(f) &= \{L(\text{M}) : \text{M is a deterministic Turing machine acceptor} \\
&\qquad \text{which operates within time bound} f\} \\
\text{NTIME}(f) &= \{L(\text{M}) : \text{M is a nondeterministic Turing machine acceptor} \\
&\qquad \text{which operates within time bound} f\}.
\end{aligned}$$

We concentrate on the following complexity classes:

$$\begin{aligned}
\mathrm{P} &= \bigcup\{\mathrm{DTIME}(p) : p \text{ is a polynomial}\} \\
\mathrm{NP} &= \bigcup\{\mathrm{NTIME}(p) : p \text{ is a polynomial}\} \\
\mathrm{E} &= \bigcup_{c=1}^{\infty} \mathrm{DTIME}(2^{cn}) \\
\mathrm{NE} &= \bigcup_{c=1}^{\infty} \mathrm{NTIME}(2^{cn}) \\
\mathrm{TIME}(2^{poly}) &= \bigcup\{\mathrm{DTIME}(2^p) : \text{ is a polynomial}\}.
\end{aligned}$$

$\mathrm{TIME}(2^{poly})$ is important since it is the smallest known deterministic complexity class to contain NP.

We may specify a complexity class C as follows: Let $\{M_i\}_{i\in\mathrm{N}}$ be the standard enumeration of Turing transducers of some type, so that for every recursive set A there exists an M_i such that $A = L(M_i)$. Let $\{f_i\}_{i\in\mathrm{N}}$ be a recursively indexed set of unbounded fully time-constructible functions such that if $f_{i_1}, f_{i_2} \in \{f_i\}$ and $i_1 < i_2$, then $f_{i_1}(n) < f_{i_2}(n)$ for almost all n. Let $L(i,n) = \{x | M_i \text{ accepts } x \text{ in } f_n(|x|) \text{ steps }\}$ and define the time complexity class C to be $\{L(i,n) | i,n \in \mathrm{N}\}$. $\mathcal{F}_C$ is the class of functions computable by deterministic transducers with the same enumeration of time bounds. By choosing appropriate enumerations of classes of machines and classes of functions P, NP, E, NE, and $\mathrm{TIME}(2^{poly})$ can be specified in this fashion. We may define space classes in a similar fashion.

An *oracle Turing machine* M is a Turing machine acceptor with a distinguished oracle tape and three special states: QUERY, YES, NO. When M enters the QUERY state the next state is YES or NO depending on whether or not the string currently written on the oracle tape belongs to an external set — the *oracle set.* The oracle tape is instantly erased after this transition. M^A denotes the oracle Turing machine M with A as its oracle set, and $L(\mathrm{M}^A)$ denotes the set of strings *accepted by M relative to the set A.* Similar to above, we may enumerate the oracle Turing machines, and define *relativized* complexity classes. For example, P^A is the class of languages accepted by deterministic polynomial time-bounded oracle Turing machines with oracle A.

We fix enumerations $\{P_i\}_{i\in\mathrm{N}}$ ($\{NP_i\}_{i\in\mathrm{N}}$) of polynomial time-bounded deterministic (nondeterministic) oracle Turing machines. P_i^X and NP_i^X denote query machines

with oracle X, and $p_i(n) \doteq n^i + i$ is a strict upper bound on the length of any computation by P_i (or NP_i) with oracle X.

2.3 Reducibilities

A reducibility is a binary relation between two computable sets. In general, given two reducibilities $\leq_r$ and $\leq_s$, we say that $\leq_r$ is *weaker* than $\leq_s$ if for all computable sets A and B, $A \leq_s B$ implies $A \leq_r B$. $\leq_r$ is *properly weaker* than $\leq_s$ if $\leq_r$ is weaker than $\leq_s$ but not *vice versa*. $A \equiv_r B$ just in the case $A \leq_r B$ and $B \leq_r A$, and $A \mid_r B$ just in the case $A \not\leq_r B$ and $B \not\leq_r A$. $\leq_r$ *stratifies* $\leq_s$ if there exist computable sets A and B such that $A \equiv_s B$ and $A \mid_r B$.

A set A is *many-one reducible to a set B in polynomial time* ($A \leq_m^P B$) if there exists a polynomial time computable function f such that $x \in A$ if and only if $f(x) \in B$. A set A is *Turing reducible to a set B in polynomial time* ($A \leq_T^P B$) if and only if there is a polynomial time-bounded oracle Turing machine M such that $A = L(\mathrm{M}^B)$.

For a complexity class C, a set B is *C-hard* if for every set A in C, $A \leq_T^P B$. Note that B does not have to be an element of C. A set B is *C-complete* if $B \in C$ and, for all $A \in C$, $A \leq_m^P B$. The standard example of an NP-complete set is SAT — the set of satisfiable propositional formulas.

3 HIERARCHIES OF ALMOST EVERYWHERE COMPLEX SETS

3.1 Introduction

Many of the early theoretical results in computational complexity — the work of Hartmanis and Stearns [36], Hartmanis, Lewis and Stearns [35], and Rabin [55], for example — dealt with the construction of arbitrary difficult-to-compute recursive functions. Although all of these constructions involved diagonalization arguments, two fundamentally different concepts of "complex function" emerged. Hartmanis and Stearns constructed 0,1-valued recursive functions that required large running times for infinitely many inputs. The 0,1-valued recursive functions constructed by Rabin, on the other hand, required large running times for all but finitely many inputs.

As Gill and Blum [31] noted, the distinction between these two concepts was obscured in the intervening years due in part to the concern of classifying recursive functions by complexity classes. Recall that a recursive function f is defined to belong to the complexity class determined by a time bound h if some program computed $f(x)$ in time $h(|x|)$ for all but finitely many x. So a function is considered difficult to compute with respect to h if every program for that function took more than $h(|x|)$ steps for infinitely many inputs x.

The perceived unnaturalness of a.e. complexity also played a role in this obfuscation. There is no known natural example of a recursive function f requiring $h(|x|)$ steps to compute $f(x)$ for almost all x, where h is any reasonably large time bound. The existence of natural problems which require exponential computation time for infinitely many inputs is well documented by Meyer and Stockmeyer [50] and Ferrante and Rackoff [24].

Gill and Blum compared these two notions and attempted to provide a theoreti-

cal explanation for the fact that it appeared much harder to construct natural almost everywhere (a.e.) complex functions than natural infinitely often (i.o.) complex functions. They showed that it is impossible to prove every a.e. complex recursive function to be difficult to compute, but they interpreted a result of Landweber and Robertson [46] as showing that every i.o. complex recursive function could be proven to be i.o. complex. They concluded that it was fundamentally more difficult to construct a.e. complex functions than i.o. complex functions.

Lynch [47] proved that if a recursive set A is not solvable in polynomial time, then there is an infinite recursive set X, called A's *complexity core*, such that for every deterministic Turing machine M that recognizes A and for every polynomial p, M executes more than p steps on all but finitely many of the elements of X. Therefore, some notion of a.e. complexity is precisely what makes sets intractable. Furthermore, she showed that sets which are polynomial time many-one reducible to arbitrarily complex sets are polynomial time computable. She cited this counterintuitive result as further evidence of the unnaturalness of a.e. complexity.

Complexity cores have played a key role in the study of the structural properties of intractable sets. Even, Selman and Yacobi [23] have shown that for any recursive set not in P, it is possible to construct infinite recursive complexity core inside the set. They proved similar results in a general setting to examine what abstract properties a complexity class need have to prove the existence of recursive complexity cores. Orponen and Schöning [51] showed that recursive intractable sets have infinite cores that can be recognized in subexponetial time. They also showed that if P $\neq$ NP, then NP-complete sets have nonsparse complexity cores recognizable in subexponential time. Balcázar and Schöning [8] pointed out the strong connection between P-immunity and complexity cores.

In this chapter we first present the recursion theoretic notions of immunity and bi-immunity and show how they relate to the complexity theoretic notion of a.e. complexity. This work is perfunctory, and many of the results can be found in Balcázar and Schöning [8]. We then prove the existence of a very strong deterministic time hierarchy — a hierarchy of a.e. complex recursive sets — that is as tight as the Hartmanis-Stearns Hierarchy Theorem. This result shows that a.e. complex sets can be constructed as easily as i.o. complex sets. Also, this result is a significant improve-

ment over all previously known results for the immune and a.e. complex sets. We derive, as a corollary, a similar deterministic space hierarchy result due to Meyer and McCreight [49]. The results presented in this section are the result of a collaboration with D. Huynh (Geske and Huynh [30]).

Many sets are not a.e. complex for trivial reasons. We look at alternative ways of viewing this and present a generalized form of a.e. complexity in Section 3.4. We show that the very strong deterministic time hierarchy theorem also holds for this generalized form. Finally, in section 3.5 we apply these results to answer several questions concerning P-printable sets and generalized Kolmorgorov complexity.

3.2 Immunity and Almost Everywhere Complexity

Various researchers have derived hardness notions for computable sets. One of the earliest, as cited in the previous section, is Rabin's notion of the a.e. complex set. We generalize that idea here.

Definition 3.1 *Let C be a complexity class. A set A is C-complex if for every Turing machine M that accepts A and every function $f \in \mathcal{F}_C$, $T_M(x) > f(|x|)$ for all but finitely many $x \in \Sigma^*$*

Flajolet and Steyaert [26], and independently Constable [17], transformed the recursion theoretic notion of an immune set into a computational complexity framework. The bi-immune sets were first defined by Balcázar and Schöning [8].

Definition 3.2 *Let C be a complexity class*:

- *A set is C-immune if and only if it is infinite and has no infinite subset that belongs to C.*
- *An infinite co-infinite set A is C-bi-immune if and only if A is C-immune and $\overline{A}$ is C-immune.*

If the terms "a.e. complex", "immune", and "bi-immune" are used without reference to an underlying complexity class the class P is implicitly assumed, *e.g.*, A is immune if and only if A is P-immune.

Classes C_1 and C_2 are *strongly separated* if $C_1 \subseteq C_2$ and C_2 contains a C_1-immune set. Classes C_1 and C_2 are *very strongly separated* if $C_1 \subseteq C_2$ and C_2 contains a C_1-complex set.

Lynch [47] introduced the notion of a complexity core.

Definition 3.3 *Let C be a complexity class and $A \subseteq \Sigma^*$. An infinite set X is a C-complexity core of A [denot. C-core(A)] if for every Turing machine M that accepts A and every $f \in \mathcal{F}_C$ there are at most finitely many $x \in X$ such that $T_M(x) \leq f(|x|)$.*

The relationship between these three definitions is easy to see.

Proposition 3.4 *Let A be a recursive set, C a complexity class. The following are equivalent:*

(a) A is C-complex.

(b) A is C-bi-immune.

(c) Σ^* is a C-complexity core of A.

Proof. (c) $\Rightarrow$ (a) Obvious.

(a) $\Rightarrow$ (b) Assume the contrary, i.e., A is C-complex and not C-bi-immune. Either A is not C-immune, or $\overline{A}$ is not C-immune. The cases are symmetrical; we consider the first case only.

If A is not C-immune, then there exists an infinite subset X of A and an $f(n)$ time-bounded Turing machine M_X that recognizes X, where $f \in \mathcal{F}_C$. Let M be a Turing machine recognizer of A and construct a Turing machine M' as follows: On input x run M and M_X in parallel, accepting x as soon as M or M_X accepts x. It is easy to see that $L(M') = A$, but also for all $x \in X$, $T_{M'}(x) \leq f(|x|)$. X is infinite and $f \in \mathcal{F}_C$ so A is not C-complex, and we arrive at a contradiction.

(b) $\Rightarrow$ (c) Assume Σ^* is not a C-core of A, then there exists a Turing machine M that accepts A and there exists an $f \in \mathcal{F}_C$ such that $T_M(x) < f(|x|)$ for infinitely many x in Σ^*. Construct a Turing machine M' as follows: On input x simulate M for $f(|x|)$ steps. If M accepts x, then M' accepts x. If M rejects x, then M' rejects x. Either $L(M') = \{x : M \text{ accepts } x \text{ in } f(|x|) \text{ steps}\}$ is infinite, or $\{x : M \text{ rejects } x \text{ in } f(|x|) \text{ steps}\}$ is infinite. Both sets are in C; one set is a subset of A; the other set is a subset of $\overline{A}$. Therefore, we arrive at a contradiction.$\square$

We will find the following characterization of the C-complex sets to be useful.

Theorem 3.5 (Balcázar and Schöning [8]) *A recursive set A is C-complex if and only if for every function $f \in \mathcal{F}_C$ such that $f(A) \cap f(\overline{A}) = \emptyset$, f is finite-one.*

Proof. Assume that a set A is not C-complex. Then there exists an infinite subset A_1 of A (or of $\overline{A}$) such that $A_1 \in C$. Define $f : \Sigma^* \to \Sigma^*$ by

$$f(x) = \begin{cases} y & \text{for all } x \in A_1; \\ x & \text{otherwise,} \end{cases}$$

where y is some fixed string in A_1. Clearly $f(A) \cap f(\overline{A}) = \emptyset$, $f \in \mathcal{F}_C$, and f is not finite-one.

Conversely, assume there exists an $f \in \mathcal{F}_C$, $f(A) \cap f(\overline{A}) = \emptyset$ and f is not finite-one. Then there exists a $y \in \Sigma^*$ such that $f^{-1}(y) \subseteq A$ is infinite, $f^{-1}(y) \in C$ and so, A is not C-complex. $\square$

3.3 Hierarchies of Almost Everywhere Complex Sets

One of the earliest results in computational complexity was Hartmanis and Stearns' hierarchy theorem for deterministic time classes [36]. The recursive functions constructed in that paper required large running times for infinitely many inputs. Gill and Blum [31] compared this notion of i.o. complexity to the stronger notion of a.e. complexity formulated by Rabin [55], and they concluded that "it may be fundamentally more difficult to construct almost everywhere complex functions than infinitely often complex functions." Contrary to this intuition, in this section we prove the existence of a very strong deterministic time hierarchy — a hierarchy of a.e. complex sets — that is as tight as the infinitely often case (although in a somewhat less general form).

This result is a significant improvement over the only previously known result for a.e. complex sets, as reported by Seiferas, Fischer and Meyer [62]. In fact, this result is a significant improvement over the hierarchy theorem for immune sets presented by Flajolet and Steyaert [27]. As a corollary we obtain the deterministic space hierarchy result due to Meyer and McCreight [49].

The results presented here are independent of the results of Paul [52] and Fürer [28]; they present much tighter time hierarchies of i.o. complex sets than the Hartmanis-Stearns result by restricting the class of machines examined to a fixed number of worktapes. We are interested in a.e. complexity and place no such restriction on the machines examined here, and so our results are the best possible using current simulation techniques.

The intuition behind the very strong time hierarchy theorem is that if we are able to "slow down" the diagonalization process of the Hierarchy Theorem sufficiently, it should be possible to ensure that the constructed set is a.e. complex. In fact for many classes, it is possible to slow down this diagonalization by an arbitrarily small function — the function $f(n)$ given below.

Theorem 3.6 *If $t_2(n)$ is a monotone increasing fully time-constructible function such that*

$$\liminf_{n\to\infty} \frac{t_1(n)\log t_1(n)}{t_2(n)} = 0,$$

and there exists a fully time-constructible monotone increasing and unbounded function $f(n)$ such that

$$\liminf_{n\to\infty} \frac{f(n)t_1(n)\log t_1(n)}{t_2(n)} = 0,$$

then there exists a DTIME$(t_1(n))$*-complex set in* DTIME$(t_2(n))$.

Example 3.7 *If $t_1(n) = 2^n$, and $t_2(n) = n^2 2^n$, then by the Hartmanis-Stearns Hierarchy Theorem,* DTIME$(2^n) \subset$ DTIME$(n^2 2^n)$. *By Theorem 3.6, letting $f(n) = \log n$, there exists a* DTIME(2^n)*-complex set in* DTIME$(n^2 2^n)$.

The best previously known result [62] was that there existed DTIME(2^n)-complex sets in DTIME$((2+\epsilon)^n)$, for any $\epsilon > 0$. Note, as in this example, that for many running times $t_1(n)$ and $t_2(n)$ that satisfy the condition that $\lim_{n\to\infty} \inf t_1(n)\log t_1(n)/t_2(n) = 0$, a suitable $f(n)$ such that the condition $\lim_{n\to\infty} \inf f(n)t_1(n)\log t_1(n)/t_2(n) = 0$ is satisfied can be found.

Example 3.8 *There exist* P*-complex sets in* DTIME$(n^{\log n})$.

The construction of an a.e. complex set is based on a finite-injury priority argument. Given functions $t_2(n)$ and $f(n)$, we construct a recursive a.e. complex set A inductively on the enumerated strings of Σ^*. We consider the following infinite enumeration of *restraining conditions*:

$$R_i : \varphi_i \text{ is not finite} - \text{one} \Rightarrow \varphi_i(A) \cap \varphi_i(\overline{A}) \neq \emptyset.$$

A condition R_j is *satisfiable* if its antecedent is true. A satisfiable condition R_j is *satisfied* if $\varphi_j(A) \cap \varphi_j(\overline{A}) \neq \emptyset$.

At each stage we attempt to satisfy the least (smallest indexed) restraining condition that has not yet been satisfied. Such a construction is given in Figure 3.1. In the construction, S is a set of restraining conditions (transducer indices) to be considered.

Lemma 3.9 $A \in \text{DTIME}(t_2(n))$.

Proof. We show that there exists a Turing machine that executes the algorithm given in Figure 3.1 in time $t_2(n)$. On an input x of length n, the time to cancel machines from previous stages requires time $t_2(n)$. At most $\lfloor\sqrt{f(n)}\rfloor$ transducers must be simulated on at most $\lfloor\sqrt{f(n)}\rfloor$ strings, so at most $f(n)$ simulations must be made. Each simulation is run within $\lfloor t_2(n)/f(n)\rfloor$ steps. (Note that for a machine to do this requires that both $t_2(n)$ and $f(n)$ be fully time-constructible.) Therefore the simulations require at most $t_2(n)$ steps, and the total time necessary to execute the algorithm is $O(t_2(n))$. By the conditions placed on $t_1(n)$ and $t_2(n)$, we have $\lim_{n\to\infty} \inf t_2(n)/n = \infty$ and so, by the Linear Speedup Theorem, we arrive at our desired result.□

Lemma 3.10 *For every $t_1(n)$ computable φ_j that is not finite-one, R_j is satisfied.*

Proof. Let T_l be a $t_1(n)$ time-bounded transducer such that φ_l is not finite-one and $\varphi_l(A) \cap \varphi_l(\overline{A}) = \emptyset$. Because of the conditions placed on $t_2(n)$ and $f(n)$, we can assume without loss of generality that at some stage i in the construction, for all indicies $j < l$, either $j \notin S$, *i.e.*, R_j has already been satisfied, or φ_j is finite-one. So, l is the smallest satisfiable index in S at stage i, and remains the smallest satisfiable index at all future stages until R_l is satisfied.

```
stage i
   begin
      Let n := ⌊log i⌋;
      Let m := min(f(n), ⌊log n⌋);
      Let S := {1, ..., ⌊√m⌋};
      Let W := set of the first ⌊√m⌋ strings;
      Within time t2(n) execute as many previous stages as possible,
         beginning with stage 1, removing transducer indices out of S
         that have been diagonalized during previous stages;
      for each j ∈ S do
         Simulate Tj for ⌊t2(n)/f(n)⌋ steps to determine whether there
            is a string w in W such that Tj(w) = Tj(wi);
         if yes
         then begin
            Let ĵ be the least transducer index with this property;
            Let wi be the least string in W with this property;
            if wi ∈ A
            then add wi to Ā
            else add wi to A
         end
   end stage i
```

Figure 3.1: Inductive construction of a complex set

The time required to simulate T_l on an input of length n is $c\,t_1(n)\log t_1(n)$, where c is a constant depending on T_l. Note that the Hennie-Stearns Theorem is needed here since we make no assumptions about the number of worktapes T_l may have. We are only allowed to simulate T_l for $\lfloor t_2(n)/f(n)\rfloor$ steps. However, since φ_l is not finite-one, and by the conditions placed on $t_2(n)$ and $f(n)$, there exist strings w_j, w_k such that $|w_j| < \left\lfloor\sqrt{\min(f(|w_k|), \lfloor\log|w_k|\rfloor)}\right\rfloor$, $\varphi_l(w_j) = \varphi_l(w_k)$, and w_k is a sufficiently large string such that $c\,t_1(|w_j|)\log t_1(|w_j|) + c\,t_1(|w_k|)\log t_1(|w_k|) < t_2(|w_k|)/f(|w_k|)$. Therefore there is enough time to complete the simulation and witness the fact that $\varphi_l(w_j) = \varphi_l(w_k)$. If $w_j \in A$, then $w_k \in \overline{A}$, otherwise $w_j \in \overline{A}$ and $w_k \in A$. In either case $\varphi_l(A) \cap \varphi_l(\overline{A}) \neq \emptyset$ and R_l is satisfied.□

Proof of Theorem 3.6. That A is a DTIME($t_1(n)$)-complex set in DTIME($t_2(n)$) follows directly from Lemma 3.9 and the fact (easily verifiable) that if Lemma 3.10 holds then the conditions of Theorem 3.5 are satisfied.□

Since we are allowed to reuse tape cells, it is not necessary to slow down the diagonalization as was necessary for the time case, and we derive the tight space hierarchy theorem of Meyer and McCreight. One should also note here that, as with the time case, that $s_2(n)$ must be monotone increasing to insure the cancellation of transducers diagonalized at previous stages.

Corollary 3.11 *If $s_2(n)$ is a fully space-constructible monotone increasing function, $s_1(n)$ and $s_2(n)$ are at least $\log_2 n$, and*

$$\lim_{n\to\infty} \inf \frac{s_1(n)}{s_2(n)} = 0,$$

then there is a DSPACE($s_1(n)$)*-complex set in* DSPACE($s_2(n)$).

As the proof of Corollary 3.11 is very similar to that of Theorem 3.6, we only present the inductive construction of an a.e. complex set in Figure 3.2 and leave to the reader the details of verification.

The very strong hierarchy theorems for deterministic time and space are as tight as the hierarchies for the i.o. cases. Traditionally, translational lemmas have been used to derive even tighter results than allowable by the hierarchy theorems. For example, DTIME(2^n) $\subset$ DTIME($n\,2^n$) can only be obtained in this fashion. Unfortunately, the

```
stage i
  begin
    Let n := ⌊log i⌋;
    Let S := {1, ..., ⌊log log n⌋};
    Let W := set of the first ⌊log log n⌋ strings;
    Within space s_2(n) execute as many previous stages as possible,
      beginning with stage 1, removing transducer indices out of S
      that have been diagonalized during previous stages;
    for each j ∈ S do
      Within space s_2(n) simulate T_j to determine whether there is
        a string w in W such that T_j(w) = T_j(w_i);
      if yes
      then begin
        Let ĵ be the least transducer index with this property;
        Let w_i be the least string in W with this property;
        if w_i ∈ A
        then add w_i to Ā
        else add w_i to A
      end
  end stage i
```

Figure 3.2: Space preserving inductive construction of a complex set

padding technique used in translation lemmas (See for example Hopcroft and Ullman [39]) does not carry over for the a.e. complex case; padded strings allow for easily recognized subsets to exist. As padding plays a key role in Cook's [19] and Seiferas, Fischer, and Meyer's [62] derivation of hierarchies for the nondeterministic classes, it appears that new techniques will have to be developed for results of this nature to be achieved for the a.e. complex cases. Resolution of these difficulties remain open.

3.4 Generalized Notions of Immunity

The complexity of set recognition, as pointed out by Lynch, lies in the recognition of a set's complexity core. The fact that a set A is a.e. complex precisely when Σ^*, the entire domain of A, is a complexity core of A emphasizes the restrictiveness of a.e. complexity. Frequently the syntactic structure of a set makes it recognizable on many of its inputs. For example, there exists a Turing machine M that runs in real-time such that $\overline{L(M)} = \{1\}^*$. Therefore for every tally language A, $L(M) \subseteq \overline{A}$. This fact says nothing about the complexity of a given tally set; tally sets of arbitrary complexity can easily be constructed.

To overcome this difficulty, alternative notions of immunity have been developed. Orponen and Schöning [51] considered sets that are uniformly hard to decide everywhere except on a single easily recognized subset. They called these sets *almost* P-*immune*, that is, a set is almost P-immune if it is the disjoint union of a P-immune set and a set in P. Grollmann and Selman [32] considered the notion of partial immunity: A non-sparse set S is *partially immune* to complexity class C if every subset of S in C is a sparse set. We define our own generalized notion of immunity and bi-immunity.

Definition 3.12 *Let C be a complexity class and $\Gamma \subseteq \Sigma^*$:*

- *A set A is $C|\Gamma$-immune if and only if $A \cap \Gamma$ is infinite and every infinite subset of A in C has a finite intersection with Γ.*
- *An infinite co-infinite set A is $C|\Gamma$-bi-immune if and only if A is $C|\Gamma$-immune and $\overline{A}$ is $C|\Gamma$-immune.*

So, an infinite set A is $C|\Gamma$-bi-immune if no infinite subset of $A \cap \Gamma$ is in C and no infinite subset of $\overline{A} \cap \Gamma$ is in C. For the case of tally languages, by letting $\Gamma = \{1\}^*$ this generalized notion of a.e. complexity removes the trivial instances — those instances containing a zero — from consideration. Note that if $\Gamma = \Sigma^*$, then this definition is precisely Definition 3.2. There does not appear to be any relation between this version of immunity and those of Orponen and Schöning and Grollmann and Selman. Our notion is perhaps most closely related to Ambose-Spies' notion of a sub-problem[5]. In a similar fashion, we may generalize the notion of a C-complex set.

Definition 3.13 *Let C be a complexity class and $\Gamma \subseteq \Sigma^*$. A set A is $C|\Gamma$-complex if for every Turing machine M that accepts A and every function $f \in \mathcal{F}_C$, $T_M(x) > f(|x|)$ for all but finitely many $x \in \Gamma$.*

The "naturalness" of our generalization can be seen in the following result.

Proposition 3.14 *Let A be a recursive set, C a complexity class and Γ an infinite subset of Σ^*. The following are equivalent:*

(a) A is $C|\Gamma$-complex.

(b) A is $C|\Gamma$-bi-immune.

(c) Γ is a C-complexity core of A.

Proof. (c) $\Rightarrow$ (a) Obvious.

(a) $\Rightarrow$ (b) Assume the contrary, i.e., A is $C|\Gamma$-complex and not $C|\Gamma$-bi-immune. Either A is not $C|\Gamma$-immune, or $\overline{A}$ is not $C|\Gamma$-immune. The cases are symmetrical; we consider the first case only.

If A is not $C \mid \Gamma$-immune, then there exists an infinite subset $X \cap \Gamma$ of A and an $f(n)$ time-bounded Turing machine M_X that recognizes X, where $f \in \mathcal{F}_C$. Let M be a Turing machine recognizer of A and construct a Turing machine M' as follows: On input x run M and M_X in parallel, accepting x as soon as M or M_X accepts x. It is easy to see that $L(M') = A$, but also for all $x \in X$, $T_{M'}(x) \leq f(|x|)$. $X \cap \Gamma$ is infinite and $f \in \mathcal{F}_C$ so A is not $C|\Gamma$-complex, and we arrive at a contradiction.

(b) $\Rightarrow$ (c) Assume Γ is not a C-core of A, then there exists a Turing machine M that accepts A and there exists an $f \in \mathcal{F}_C$ such that $T_M(x) < f(|x|)$ for infinitely many x in Γ. Construct a Turing machine M' as follows: On input x simulate M

for $f(|x|)$ steps. If M accepts x, then M' accepts x. If M rejects x, then M' rejects x. $L(M') \in \mathcal{C}$ and $L(M') \subseteq A$. Assume $L(M')$ is infinite, then clearly $L(M')$ has an infinite intersection with Γ and we have a contradiction. If we assume $L(M')$ is finite, then $\overline{L(M')}$ is infinite and the result is the same *mutatis mutandis.*$\square$

Indeed, the results of the previous sections can be translated into results about generalized a.e. complexity.

Theorem 3.15 *A recursive set A is $\mathcal{C}|\Gamma$-complex if and only if for every function $f \in \mathcal{F}_{\mathcal{C}}$ such that $f(A) \cap f(\overline{A}) = \emptyset$, $f^{-1}(y) \cap \Gamma$ is finite for every $y \in \Sigma^*$.*

Proof. Assume that a set A is not $\mathcal{C}|\Gamma$-complex, then there exists an infinite subset A_1 of A (or $\overline{A}$) such that $A_1 \in \mathcal{C}$ and $A_1 \cap \Gamma$ is infinite. Define $f : \Sigma^* \to \Sigma^*$ by

$$f(x) = \begin{cases} y & \text{for all } x \in A_1; \\ x & \text{otherwise,} \end{cases}$$

where y is some fixed string in A_1. $f(A) \cap f(\overline{A}) = \emptyset$, $f \in \mathcal{F}_{\mathcal{C}}$ and there exists a $y \in \Sigma^*$ such that $f^{-1}(y) \cap \Gamma$ is infinite.

Conversely, assume there exists an $f \in \mathcal{F}_{\mathcal{C}}$, $f(A) \cap f(\overline{A}) = \emptyset$ and a $y \in \Sigma^*$ such that $f^{-1}(y) \cap \Gamma$ is infinite. Either $f^{-1}(y) \subseteq A$, or $f^{-1}(y) \subseteq \overline{A}$, $f^{-1}(y) \in \mathcal{C}$ and so, A is not $\mathcal{C}|\Gamma$-complex.$\square$

Given a relatively easy to recognize subset Γ of Σ^*, we can restrict our attention to this subset and construct new sets with complexity cores contained in this subset. Using the techniques of Theorem 3.6, these new sets can be made to be a.e. complex in our generalized sense. The proof of Theorem 3.16 is similar to that of Theorem 3.6; for the sake of completeness, we give the proof in detail.

Theorem 3.16 *Let Γ be an infinite subset of Σ^* in $\mathrm{DTIME}(t_2(n))$. If $t_2(n)$ is a monotone increasing fully time-constructible function such that*

$$\lim_{n\to\infty} \inf \frac{t_1(n) \log t_1(n)}{t_2(n)} = 0,$$

and there exists a fully time-constructible monotone increasing and unbounded function $f(n)$ such that

$$\lim_{n\to\infty} \inf \frac{f(n) t_1(n) \log t_1(n)}{t_2(n)} = 0,$$

then there exists a subset of Γ in $\mathrm{DTIME}(t_2(n))$ that is $\mathrm{DTIME}(t_1(n))|\Gamma$-complex.

```
stage i
  begin
    Let n := ⌊log i⌋;
    Let m := min(f(n), ⌊log n⌋);
    Let S := {1, ..., ⌊√m⌋};
    Let W := set of the first ⌊√m⌋ strings;
    Within time t2(n) determine if w_i ∈ Γ. If not, then exit stage i;
    Within time t2(n) execute as many previous stages as possible,
      beginning with stage 1, removing transducer indices out of S
      that have been diagonalized during previous stages;
    for each j ∈ S do
      Simulate T_j for ⌊t2(n)/f(n)⌋ steps to determine whether there
        is a string w in W ∩ Γ such that T_j(w) = T_j(w_i);
      if yes
      then begin
        Let ĵ be the least transducer index with this property;
        Let w_i be the least string in W with this property;
        if w_i ∈ A
        then add w_i to Ā
        else add w_i to A
      end
  end stage i
```

Figure 3.3: Inductive construction of a DTIME($t_1(n)$)|Γ-complex set

We construct a DTIME($t_1(n)$)|Γ-complex set inductively *via* a finite-injury priority argument. By virtue of Theorem 3.15, we only need concern ourselves with the following enumeration of restraining conditions:

$$R_i : \exists y\, \varphi_i^{-1}(y) \cap \Gamma \text{ is not finite} \Rightarrow \varphi_i(A) \cap \varphi_i(\overline{A}) \neq \emptyset.$$

It is only necessary, therefore, to consider strings in Γ; this can be determined "quickly." The construction is given in Figure 3.3. Again, S is the set of restraining conditions to be considered.

Lemma 3.17 $A \in \text{DTIME}(t_2(n))$.

Proof. We show that there exists a Turing machine that executes the algorithm given in Figure 3.1 in time $t_2(n)$. On an input x of length n, the time to determine whether $x \in \Gamma$ requires time $t_2(n)$. The time to cancel machines from previous stages requires time $t_2(n)$. At most $\lfloor\sqrt{f(n)}\rfloor$ transducers must be simulated on at most $\lfloor\sqrt{f(n)}\rfloor$ strings, so at most $f(n)$ simulations must be made. Each simulation is run within $\lfloor t_2(n)/f(n)\rfloor$ steps. Therefore the simulations require at most $t_2(n)$ steps, and the total time necessary to execute the algorithm is $O(t_2(n))$. By the conditions placed on $t_1(n)$ and $t_2(n)$, we have $\lim_{n\to\infty} \inf t_2(n)/n = \infty$ and so, by the Linear Speedup Theorem, we arrive at our desired result.□

Lemma 3.18 *For every $t_1(n)$ computable φ_j such that $\varphi_j^{-1}(y) \cap \Gamma$ is not finite for some $y \in \Sigma^*$, R_j is satisfied.*

Proof. Let T_l be a $t_1(n)$ time-bounded transducer such that R_l is satisfiable. Because of the conditions placed on $t_2(n)$ and $f(n)$, we can assume without loss of generality that at some stage i in the construction, for all indicies $j < l$, either $j \notin S$, *i.e.*, R_j has already been satisfied, or R_j is not satisfiable. So, l is the smallest satisfiable index in S at stage i, and remains the smallest satisfiable index at all future stages until R_l is satisfied.

The time required to simulate T_l on an input of length n is $c\,t_1(n)\log t_1(n)$, where c is a constant depending on T_l. We are only allowed to simulate T_l for $\lfloor t_2(n)/f(n)\rfloor$ steps. However, since $\varphi_l^{-1}(y) \cap \Gamma$ is not finite for some $y \in \Sigma^*$, and by the conditions placed on $t_2(n)$ and $f(n)$, there exist strings w_j, w_k in Γ such that $|w_j| < \left\lfloor\sqrt{\min(f(|w_k|), \lfloor\log|w_k|\rfloor)}\right\rfloor, \varphi_l(w_j) = \varphi_l(w_k)$, and w_k is a sufficiently large string such that $c\,t_1(|w_j|)\log t_1(|w_j|) + c\,t_1(|w_k|)\log t_1(|w_k|) < t_2(|w_k|)/f(|w_k|)$. Therefore there is enough time to complete the simulation and witness the fact that $\varphi_l(w_j) = \varphi_l(w_k)$. If $w_j \in A$ then $w_k \in \overline{A}$, otherwise $w_j \in \overline{A}$ and $w_k \in A$. In either case $\varphi_l(A) \cap \varphi_l(\overline{A}) \neq \emptyset$ and R_l is satisfied.□

Proof of Theorem 3.16. Follows directly from Lemmas 3.17 and the fact Lemma 3.10 ensures that the conditions of Theorem 3.5 are satisfied.□

3.5 Applications

Generalized Kolmogorov complexity, introduced by Hartmanis [33] as a method for studying the amount of information contained in individual strings, has generated considerable interest. See for example the work by Allender [3] and Ko, Orponen, Schöning and Watanabe [43]. Informally, the *Kolmogorov complexity* of a string is the shortest program that generates it. Hartmanis' generalization includes information about not only how far a string can be compressed, but how fast it can be restored.

Definition 3.19 *For a Universal Turing machine M_U and functions g and G on the natural numbers, the generalized Kolmogorov complexity is defined as*

$$K_U[g(n), G(n)] = \{x : \exists y(|y|) < g(|x|) \text{ and } M_U(y) = x \text{ in } \leq G(|x|) \text{ steps})\}.$$

It was shown in [33] that there exists a universal Turing machine M_U such that for any other Turing machine M_V there exists a c_0 such that $K_V[g(n), G(n)] \subseteq K_U[g(n) + c_0, c_0 G(n) \log G(n) + c_0]$.

The following propositions are straightforward.

Proposition 3.20 *Every tally set is a subset of $K_U[c \log n, n^c]$ for some integer c.*

Proposition 3.21 *For every integer c, $K_U[c \log n, n^c] \in$ P.*

The P-printable sets were first defined by Hartmanis and Yesha [37], and they arise naturally in various areas, such as P-uniform circuit complexity and data compression. Allender [3] has studied the strong connections between generalized Kolmogorov complexity and P-printable sets.

Definition 3.22 *A set S is* P*-printable if there is a polynomial time computable function which, on input 1^n, lists the elements of S which have length $\leq n$.*

Clearly every P-printable set is sparse and in P.

Proposition 3.23 (Allender and Rubinstein [4]) *S is p-printable if and only if S is p-isomorphic to a tally language in P.*

We briefly examine these concepts here; specifically, we examine what our notion of generalized almost everywhere complexity can say about these concepts. The following corollaries follow directly from Theorem 3.16.

Corollary 3.24 *There exists* $P|\{1\}^*$*-complex sets in* $DTIME(n^{\log n})$.

Corollary 3.25 *There exist infinite tally sets in* $DTIME(n^{\log n})$ *that have no infinite* P*-printable subsets.*

Proof. By Theorem 3.16 we can construct an infinite tally set $A \in DTIME(n^{\log n})$ that is $P|\{1\}^*$-complex. Assume $S \subseteq A$ is an infinite P-printable set. This implies that S is in P, but then there exists an infinite tally subset of A in P. This contradicts the fact that A is $P|\{1\}^*$-complex.□

Theorem 3.26 *There exists a sparse set* A *in* $DTIME(n^{\log n})$ *such that no P-printable set is a subset of* A*, nor* $\overline{A}$.

Proof. Let $\Gamma = K_U[\log^2 n, n^{\log\log n}]$; $\Gamma \in DTIME(n^{\log n})$. Furthermore, $K_U[c \log n, n^c] \subseteq K_U[\log^2 n, n^{\log\log n}]$ for every integer c, and every P-printable set is a subset of Γ. Let A be a sparse $P|\Gamma$-complex set. A can be constructed such that $A \in DTIME(n^{\log n})$. By definition, no infinite set $S \subseteq K_U[\log^2 n, n^{\log\log n}]$, $S \in P$, is a subset of A, nor can S be a subset of $\overline{A}$. □

Theorem 3.27 *There exists a set* A *in* $DTIME(n^{\log n})$ *such that the only infinite subsets of* A*, or* $\overline{A}$*, in* P *are of low generalized Kolmogorov complexity.*

Proof. Let $\Upsilon = \Sigma^* - K_U[\log^2 n, n^{\log\log n}]$. Construct a set $A \in DTIME(n^{\log n})$ such that A is $P|\Upsilon$-complex. Any set in P that is a subset of A, or $\overline{A}$, must be of low generalized Kolmogorov complexity, since no infinite set S in Υ that is in P can be a subset of A or $\overline{A}$. These are precisely the sets that do not have low generalized Kolmogorov complexity.□

4 POLYNOMIAL COMPLEXITY DEGREES

4.1 Introduction

The early history of the theory of NP-complete sets was fraught with confusing attempts at notation and concepts. Several authors attempted noneffective formulations of NP-completeness. For example, Sahni [59] and Sahni and Gonzales [60] tried to formulate NP-completeness without the use of polynomial time-bounded reducibilities. They called a set $A \in$ NP P-*complete* if and only if membership of A in P implied P = NP. It soon became clear that such a nonconstructive test did not properly capture the notion of NP-completeness. For, as Book, Wrathall, Selman and Dobkin [14] pointed out, if P $\neq$ NP then every set in NP $-$ P obeys the criterion set forth by Sahni, and by Ladner's result [44] that if P $\neq$ NP then there exist a multitude of sets in NP - P that are not polynomially equivalent, it follows that P-complete is not the same as NP-complete.

Nevertheless, there is an attractiveness to a nonconstructive approach, and we pursue such an approach here. We show that the *intuition* of these early researchers was correct — though their methods were wrong. We define a binary relation that nonconstructively relates the computational complexity of two computable sets. We show that this relation is properly weaker than polynomial time Turing reducibility, and yields new completeness and hardness notions for complexity classes. This new completeness notion differs from $\leq_T^P$-complete for sets in E, and the hardness notion differs from $\leq_T^P$-hardness for sets in NP. Furthermore, through the use of this relation we show that if a complexity class C contains a set A whose "hard instances" are uniformly distributed, then every $\leq_T^P$-complete set for C must have its hard instances uniformly distributed. This uniform distribution property holds for the $\leq_T^P$-complete sets in E and TIME(2^{poly}).

```
begin
  input x;
   ⋮
  for (up to p(|x|) iterations) do
  begin
     ⋮
    generate a query q, |q| ≤ p(|x|), to oracle B;
    input q to M_B;
      run M_B on q;
    return (answer to query);
     ⋮
  end;
   ⋮
end.
```

Figure 4.1: Turing reduction of a set A to a set B

4.2 Polynomial Complexity Degrees

The importance of polynomial time bounded Turing reducibility is the fact that, placing faith in the polynomial time variant of Church's Thesis it is the most general *constructive* relation which, when it holds between two sets A and B, $A \leq_T^P B$ asserts that the "complexity" of A is "polynomially related" to the "complexity" of B. We wish to make this statement precise.

Let A and B be recursive sets. When we assert that $A \leq_T^P B$ we are asserting the existence of a polynomial time-bounded oracle Turing machine that recognizes A using B as an oracle. Consider the oracle Turing machine program given in Figure 4.1 that witnesses $A \leq_T^P B$, runs in polynomial time-bound p, and in which queries to the oracle are replaced with executions of a Turing machine M_B that recognizes B. The reduction yields a Turing machine M_A which accepts A so that

$$\forall x \in \Sigma^* \left(T_{\mathrm{M}_A}(x) \leq p(|x|) \max \{ T_{\mathrm{M}_B}(y) : |y| \leq p(|x|) \} \right). \tag{4.1}$$

We have seen from Figure 4.1 and Equation 4.1 that $A \leq_T^P B$ implies the existence

of a polynomial p such that

$$\forall \mathrm{M}_B \exists \mathrm{M}_A \forall x \in \Sigma^* (T_{\mathrm{M}_A}(x) \leq p(|x|) \max \{ T_{\mathrm{M}_B}(y) : |y| \leq p(|x|) \}). \qquad (4.2)$$

We take (4.2) to be the mathematical definition of the phrase "the complexity of A is polynomially related to the complexity of B."

Definition 4.1 *A binary relation $\leq_C$ over the recursive sets is defined as follows. Given recursive sets A and B, $A \leq_C B$ if and only if there exists a polynomial p such that*

$$\forall \mathrm{M}_B \exists \mathrm{M}_A \forall^{\infty} x \in \Sigma^* (T_{\mathrm{M}_A}(x) \leq p(|x|) \max \{ T_{\mathrm{M}_B}(y) : |y| \leq p(|x|) \}).$$

Proposition 4.2 *(i) $\leq_C$ is a reflexive and transitive relation.*

(ii) $\leq_C \cap \leq_C^{-1}$ is an equivalence relation.

The equivalence classes of this relation are called the *polynomial complexity degrees*, and for each recursive set A, the equivalence class that contains A is denoted $C(A)$, for it represents a formal definition of the "polynomial complexity" of A. Note that the approach taken here to define the polynomial complexity of A is analogous to the approach taken by mathematicians in defining the cardinality of a set A.

Definition 4.3 *A binary relation $\leq$ over the complexity degrees is defined as follows. Given complexity degrees* **a** *and* **b**,

$$\mathbf{a} \leq \mathbf{b} \Leftrightarrow \exists A \exists B (A \in \mathbf{a} \wedge B \in \mathbf{b} \wedge A \leq_C B).$$

Proposition 4.4 *Let A and B be recursive sets.*

(i) $\leq_C$ is a well defined partial ordering on the polynomial complexity degrees.

(ii) $C(A) \leq C(B) \Leftrightarrow A \leq_C B$.

(iii) $A \leq_T^P B \Rightarrow C(A) \leq C(B)$.

(iv) $\leq_C$ is an upper semi-lattice with **0***-element,* $\mathbf{0} = \mathrm{P}$.

$A \leq_C B$ is the weakest notion on the recursive sets A and B that captures all other polynomial time-bounded reducibilities; it is even weaker than the reducibility defined by Even, Long and Yacobi [21].

Definition 4.5 *Let C be a complexity class.*

- *A recursive set A is $\leq_C$-hard for C if for every recursive set B in C, $B \leq_C A$.*
- *A recursive set A is $\leq_C$-complete for C if A is $\leq_C$-hard for C and $A \in C$.*

Proposition 4.6 *Every $\leq_m^P$-complete set for a class C is $\leq_C$-complete for C.*

The proof follows directly from Proposition 4.4 (*iii*).

4.3 Basic Results

The converse of Proposition 4.4 (*iii*) fails in several strong ways. We will show (*cf.* Corollary 4.14) that if P $\neq$ NP, then there exists a recursive set A that is $\leq_C$-hard for NP, but is not $\leq_T^P$-hard for NP. Further, we show (*cf.* Theorem 4.18) that there exist recursive sets A and B in E such that $C(A) = C(B)$ and $A \not\equiv_T^P B$.

Proposition 4.7 *Let C be a complexity class. Every C-complex set is $\leq_C$-hard for C.*

Proof. Let A be a C-complex set. For an arbitrary set $B \in C$ there exists $f \in \mathcal{F}_C$ and a Turing machine M_B recognizing B such that for all x in Σ^*, $T_{M_B}(x) \leq f(|x|)$. Since A is C-complex, for every Turing machine M_A recognizing A, $T_{M_A}(x) > f(|x|)$ for all but finitely many x in Σ^*. It follows from the definition of $\leq_C$ that $B \leq_C A$. Since B was an arbitrarily chosen set in C, A is $\leq_C$-hard for C. □

Proposition 4.8 *If for some polynomial p, a set A is* DTIME$(2^{p(n)})$*-complex, then A is $\leq_C$-hard for* TIME(2^{poly}).

Proof. Let B be a set recognized by a deterministic $2^{q(n)}$ time-bounded Turing machine, for some polynomial q. Let p_1 be a polynomial such that $p(p_1(n)) > q(n)$ for sufficiently large n. Then for all strings x,

$$T_{M_B}(x) \leq 2^{q(|x|)} < 2^{p(p_1(|x|))}. \tag{4.3}$$

But note that since A is DTIME$(2^{p(n)})$-complex, for every Turing machine T_{M_A} that accepts A, and all but finitely many x,

$$\max\{T_{M_A}(y) : |y| \leq p_1(|x|)\} = \max\{T_{M_A}(y) : |y| = p_1(|x|)\} = 2^{p(p_1(|x|))}. \quad (4.4)$$

It follows from (4.3) and (4.4) that $B \leq_C A$.$\square$

These are rather appealing results, for intuitively, a set that is C-complex is computationally harder than any set in C, and $\leq_C$ captures this idea. It follows from these propositions that one way to obtain $\leq_C$-hard sets for DTIME$(t(n))$ is to use Theorem 3.6 in order to obtain DTIME$(t(n))$-complex sets. We shall see that this is not the case for polynomial time-bounded Turing reducibility.

Theorem 4.9 *Let C be a complexity class. For every recursive set $A \in C-\mathrm{P}$, there exists a sparse recursive set B such that B is C-complex and $A \not\leq_T^P B$.*

The inductive construction of the set B is based on a finite-injury priority argument. Let $\{M_i\}_{i\in\mathbf{N}}$ be an enumeration of Turing machines time-bounded by functions of $\mathcal{F}_C$. We consider two infinite enumerations of restraining conditions:

$$R_i : ||L(M_i) \cap B|| = \infty \Rightarrow L(M_i) \not\subseteq B. \quad (4.5)$$

$$\hat{R}_i : ||L(M_i) \cap \overline{B}|| = \infty \Rightarrow L(M_i) \not\subseteq \overline{B}. \quad (4.6)$$

Again, a condition R_j ($\hat{R}_j$) is satisfiable if its antecedent is true. A satisfiable condition R_j ($\hat{R}_j$) is satisfied if $L(M_j) \not\subseteq B$ ($L(M_j) \not\subseteq \overline{B}$). An added difficulty arises in satisfying that $A \not\leq_T^P B$, that is,

$$A \not\leq_T^P B \Leftrightarrow \forall i \in \mathbf{N}(A \neq L(P_i^B)). \quad (4.7)$$

The ability to satisfy (4.7) lies in the fact that if $A \notin \mathrm{P}$ and B is a finite set, then for every polynomial time-bounded deterministic oracle Turing machine P_i there must be an infinite number of witnesses to the fact that $A \neq L(P_i^B)$.

An inductive construction of the set B that satisfies these conditions is given in Figure 4.2.

Lemma 4.10 $A \not\leq_T^P B$.

stage 0
  begin
    $B := \emptyset$;
    $RSAT := \{1\}$;
    $RPSAT := \{1\}$;
    $j := 1$;
  end stage 0

stage $n (n > 0)$
  begin
    if there is an $i \in RSAT \cup RPSAT$ such that $x_n \in L(M_i)$
    then begin
      Let $\hat{i}$ be the smallest such i;
      if $\hat{i} \in RSAT$
      then $RSAT := RSAT - \{\hat{i}\}$
      else begin
        $RPSAT := RPSAT - \{\hat{i}\}$;
        $B := B \cup \{x_n\}$
      end
    end
      if $|x_n| < |x_{n+1}|$
      then
        if by running P_j with oracle $B(n)$ for $|x_n|$ steps on all inputs $\Sigma^{\leq |x_n|}$ a witness is found that $L(P_j^{B(n)}) \neq A$
        then begin
          $j := j + 1$;
          $RSAT := RSAT \cup \{j\}$;
          $RPSAT := RPSAT \cup \{j\}$;
        end
  end stage n

Figure 4.2: Inductive construction of C-complex, $A \not\leq_T^P B$ set

Proof. Assume $A \leq_T^P B$, then there exists a machine P_j in the enumeration of deterministic polynomial time-bounded oracle Turing machines such that $A = L(P_j^B)$. Note, though, that in the construction of B no more than j elements can be in B until a witness is found, if one exists, that $A \neq L(P_j^B)$. Since $A \notin P$ and B is finite, at some stage n such a witness must eventually be found. Furthermore for all strings queried of oracle B in witnessing this fact, membership in B has already been decided. Therefore, the fact that $A \notin L(P_j^B)$ will be witnessed at all future stages. This contradicts our assumption, and hence $A \not\leq_T^P B$.□

Lemma 4.11 *B is C-complex.*

Proof. Clearly if every satisfiable R_i and $\hat{R}_i$ condition is satisfied, then B will be C-complex. By Lemma 4.10 we have shown that for every integer n, P_n is successfully diagonalized, and hence conditions R_n and $\hat{R}_n$ will be considered at future stages.

Assume that for some index j that $||L(M_j)|| = \infty$ and R_j and $\hat{R}_j$ remain unsatisfied. It must be the case that for all strings x we never determine if $x \in L(M_j)$. This implies that there is always an index $i < j$ such that we determine $x \in L(M_i)$ before we can determine that $x \in L(M_j)$. This can only happen finitely often. Since $L(M_j)$ is infinite, there must exist a stage n such that $x_n \in L(M_j)$ and R_j (or $\hat{R}_j$) is the least satisfiable condition not yet satisfied, and we then satisfy R_j (or $\hat{R}_j$). Therefore our assumption is invalid, and we conclude that all satisfiable conditions R_j and $\hat{R}_j$ are satisfied.□

Proof of Theorem 4.9. Follows immediately from Figure 4.2 and Lemma 4.10 and Lemma 4.11.□

Corollary 4.12 *For every recursive set A, $A \notin$ P, there exists a sparse recursive set B such that $C(A) \leq C(B)$ and $A \not\leq_T^P B$.*

Proof. Follows directly from Theorem 4.9 and Poposition 4.7.□

Corollary 4.13 *For every set $A \in$ NP − P, there exists a sparse recursive set B such that $C(A) \leq C(B)$ and $A \not\leq_T^P B$.*

Proof. Let $A \in \text{NP} - \text{P}$. Choose a polynomial q such that $A \in \text{DTIME}(2^{q(n)})$ and let $C = \text{DTIME}(2^{q(n)})$. By Theorem 4.9 there exists a sparse recursive set B that is $\text{DTIME}(2^{q(n)})$-complex, and $A \not\leq_T^P B$. Since $\text{NP} \subseteq \text{TIME}(2^{poly})$, it follows immediately from Proposition 4.8 that $C(A) \leq C(B)$.□

If we consider the NP-complete set SAT, we immediately have the following.

Corollary 4.14 *There exists a sparse recursive set B that is $\leq_C$-hard for* NP *and is not $\leq_T^P$-hard for* NP.

This result is in contrast to Karp and Lipton [42] where it was shown that there can be no sparse NP-hard sets unless the polynomial hierarchy collapses to Σ_2^P.

By focusing our attention on the complexity class E, specifically, the a.e. complex sets in E, we can strengthen the result of Theorem 4.9 by showing that the polynomial complexity degree of the $\leq_C$-hard sets for E contain more than one $\leq_T^P$-degree.

Theorem 4.15 *For every* $\text{DTIME}(2^n)$*-complex set A in* E *there exists a sparse recursive set B in E such that $C(A) = C(B)$ and $A \not\leq_T^P B$.*

Proof. Let A be a $\text{DTIME}(2^n)$-complex set in $\text{DTIME}(2^{2n})$; such sets exist in E by use of Theorem 3.6. Let $\{M_i\}_{i \in \text{N}}$ be an effective enumeration of all deterministic 2^{2n} time-bounded Turing machines, and construct a set B via the construction given in Figure 4.2. By Theorem 4.9, B is $\text{DTIME}(2^{2n})$-complex, $A \not\leq_T^P B$ and $A \leq_C B$.

To determine whether $x_n \in B$, where $|x_n| = m$, it is sufficient to execute 2^{m+1} stages of the construction. Each stage requires $2^{O(2m)}$ steps, so the total running time is bounded by $2^{c_2 m}$ steps, for some integer constant c_2, and $B \in \text{E}$. It follows from Proposition 4.8 that $B \leq_C A$. □

Corollary 4.16 *$\leq_T^P$-complete and $\leq_C$-complete sets differ in* E.

Proof. The set $B \in \text{E}$ constructed in Theorem 4.15 is $\text{DTIME}(2^{2n})$-complex. By Proposition 4.8 B is $\leq_C$-hard for $\text{TIME}(2^{poly})$, and hence B is $\leq_C$-complete for E. B is not $\leq_T^P$-complete by construction.□

Ladner, Lynch and Selman [45] showed that $\leq_m^P$ stratifies $\leq_T^P$ in $\text{DTIME}(2^n)$. Similarly, we show that $\leq_T^P$ stratifies $\leq_C$ in E. To do this we employ the technique of

using arbitrarily large "gaps" that was used in [45]. Define the function $h : \mathbf{N} \rightarrow \mathbf{N}$ as follows:

$$\begin{aligned} h(0) &= 1, \\ h(n+1) &= 2^{h(n)}. \end{aligned}$$

Definition 4.17 *Let* $EG = \{x : |x| = h(m), \text{ for all integers } m\}$. *A set* A *has exponential gaps if* A *is a subset of* EG.

There exists a polynomial time-bounded Turing machine that on input x writes m in binary, where m is the greatest number such that $h(m) \leq |x|$, and determines if $h(m) = |x|$. Therefore, the membership question for EG is decidable in polynomial time.

Theorem 4.18 *For every* DTIME$(2^n) \mid EG$*-complex set* $A \subseteq EG$ *in* E *there exists a set* B *in* E *such that* $C(A) = C(B)$ *and* $A \mid_T^P B$.

Proof. Let $A \subseteq EG$ be a DTIME$(2^n) \mid EG$-complex set in DTIME(2^{2n}); Theorem 3.16 ensures the existence of such a set. We construct a set $B \subseteq EG$ such that B is DTIME$(2^{2n}) \mid EG$-complex and $B \in$ DTIME(2^{cn}), for some integer $c > 2$, and $A \not\leq_T^P B$ and $B \not\leq_T^P A$. The construction is similar to those of Theorem 4.9 and Theorem 4.15. Once again, we restrict our attention to the enumeration $\{M_i\}$ of 2^{2n} time-bounded deterministic Turing machines. There are two additions in the construction. First, the diagonalizations take place only on strings in EG. Secondly, we diagonalize over each machine P_i twice. The first time to ensure that $B \neq L(P_i^A)$. This step requires that we place strings into B. The second time we diagonalize over P_i is to ensure that $A \neq L(P_i^B)$. This step requires that we place strings into $\overline{B}$. Both diagonalizations take place over P_i before we examine machine P_{i+1}. The construction of B is presented in Figure 4.3.

That $B \in E$ follows along the lines of Theorem 4.15. That is, to determine whether $x_n \in B$ it is sufficient to execute $2^{|x_n+1|}$ stages of the construction. Each stage takes $2^{O(c_1|x_n|)}$ steps, plus an additional $2^{|x_n|}$ steps to simulate a machine P_j with oracle A. The remainder of the proof follows immediately from the following lemmas: Lemma 4.19; Lemma 4.20; Lemma 4.21.$\square$

stage 0
  begin
    $B := \emptyset$; $j := 1$; $k := 1$;
    $RSAT := \{1\}$; $RPSAT := \{1\}$;
  end stage 0

stage $n (n > 0)$
  begin
    if $|x_n| = h(m)$ for some integer m
    then
      if there is an $i \in RSAT \cup RPSAT$ such that $x_n \in L(M_i)$
      then begin
        Let $\hat{i}$ be the smallest such i;
        if $\hat{i} \in RSAT$
        then $RSAT := RSAT - \{\hat{i}\}$
        else begin
          $RPSAT := RPSAT - \{\hat{i}\}$;
          $B := B \cup \{x_n\}$
        end
      end
      else
        if $j = k$ **and** $p_k(h(m)) < 2^{h(m)}$ (*)
        then begin
          run machine P_k, with A as an oracle, for $p_k(|x_n|)$ steps with input x_n;
          if $x_n \notin L(P_k^A)$ **then** $B := B \cup \{x_n\}$;
          $k := k + 1$;
        end;
      if $|x_n| < |x_{n+1}|$ **and** $j < k$ (**)
      then
        if by running P_j with oracle $B(n)$ for $|x_n|$ steps on all inputs $\Sigma^{\leq |x_n|}$ a witness is found that $L(P_j^{B(n)}) \neq A$
        then begin
          $j := j + 1$;
          $RSAT := RSAT \cup \{j\}$;
          $RPSAT := RPSAT \cup \{j\}$;
        end
  end stage n

Figure 4.3: Construction of set B; $C(A) = C(B)$ and $A \mid_T^p B$

Lemma 4.19 *$A \not\leq_T^P B$ and $B \not\leq_T^P A$.*

Proof. It is sufficient to show that for every machine P_j in the enumeration of deterministic polynomial time-bounded oracle Turing machines, $A \neq L(P_j^B)$ and $B \neq L(P_j^A)$. Therefore, we diagonalize over each Turing machine P_j twice. The diagonalization to ensure $B \not\leq_T^P A$ occurs when the conditions of the if statement (*) in Figure 4.3 are satisfied; the diagonalization to ensure $A \not\leq_T^P B$ occurs when the conditions of the if statement (**) are satisfied. This second diagonalization can occur only *after* the first diagonalization since only then is the variable k incremented, thus ensuring that $j < k$. After this diagonalization occurs the variable j is incremented, once again ensuring that $j = k$, and the first diagonalization of P_{j+1} can occur. Both diagonalizations of P_j must occur before another transducer index can be added to $RSAT$ and $RPSAT$. Therefore, $\|RSAT \cup RPSAT\| \leq 2j$. That $RSAT \cup RPSAT$ remains bounded guarantees that for every positive integer j, the statement (*) is eventually reached.

For any Turing machine P_j, a stage $n = h(m)$ is eventually reached such that $p_j(n) < 2^n$. P_j is then run on input 0^n with oracle A. Note that any query q, $n < |q| < p_j(n) < 2^n$, can be answered "no" *a priori* since $A \subseteq EG$ and contains no strings in this interval. Therefore at this stage we ensure that $B \neq L(P_j^A)$. That $A \neq L(P_j^B)$ follows directly along the lines of Lemma 4.10. Once we ensure that $B \neq L(P_j^A)$ and $A \neq L(P_j^B)$ we may proceed to examine machine P_{j+1} at a future stage. Therefore we can conclude that $B \not\leq_T^P A$ and $A \not\leq_T^P B$.□

Lemma 4.20 *B is* DTIME(2^{2n})*-complex.*

Proof. Initially, $RSAT = RPSAT = \{1\}$. By Lemma 4.19 we have shown that for every integer $n \geq 1$, P_n is successfully diagonalized twice. Therefore, $n + 1$ is added to $RSAT$ and $RPSAT$, and for all integers $j \geq 1$, R_j and $\hat{R}_j$ are eventually considered. That every satisfiable R_j and $\hat{R}_j$ condition is satisfied follows along the lines of Lemma 4.11.□

Lemma 4.21 *$A \equiv_C B$.*

Proof. Let M_A be a deterministic 2^{2n} Turing recognizer of A such that

$$T_{M_A}(x) = \begin{cases} 2^{2|x|} & \text{for all } x \in EG; \\ q(|x|) & \text{otherwise,} \end{cases}$$

for some polynomial q. Since B is a DTIME(2^{2n})-complex set, for every deterministic Turing recognizer M_B of B, $T_{M_B}(x) > 2^{2|x|}$ for all but finitely many $x \in EG$. Therefore, $\forall^{\infty} x \in EG(T_{M_A}(x) < T_{M_B}(x))$, and it follows trivially that $A \leq_C B$. A similar argument shows that $B \leq_C A$. □

We will find it convenient to consider the following partion of EG:

$$EG_o = \{x : |x| = h(m), \ \textit{where } m = 1,3,5,\ldots\},$$

$$EG_e = \{x : |x| = h(m), \ \textit{where } m = 2,4,6,\ldots\}.$$

Theorem 4.22 *There exist recursive sets A and B such that $A, B \in$ DTIME($n^{\log n}$) and $A \mid_C B$.*

Proof. Let $A \subseteq EG_o$ be a P$|$ EG_o-complex in DTIME($n^{\log n}$), and $B \subseteq EG_e$ be a P$|$ EG_e-complex in DTIME($n^{\log n}$). Once again, these sets are guaranteed by Theorem 3.16. The *distance*, i.e., the difference in length, between sucessive hard instances of different lengths of A is at least a double exponential. That is, all but finitely many $x \in EG_o$ require $2^{|x|}$ steps for a Turing acceptor of A to determine membership in A, and for all y, $|x| < |y| < 2^{2^{|x|}}$, the membership question for A is trivial. But in the middle of this "gap," precisely where $|y| = 2^{|x|}$, are the hard instances of B. Therefore, for any Turing machine recognizer of B, say M_B, every Turing machine recognizer M_A of the set A, and any polynomial p we have

$$\exists^{\infty} x \, T_{M_B}(x) > p(|x|) \max\{T_{M_A}(y) : |y| \leq p(|x|)\}.$$

It follows that $B \not\leq_C A$. A similar argument shows that $A \not\leq_C B$.□

Corollary 4.23 *There exist sets A and B in* DTIME($n^{\log n}$) *such that $A \mid_T^P B$.*

4.4 The Structure of C-Complete Sets

We will now use $\leq_C$ to analyze the structure of complete sets. First we note that $\leq_C$-complete for a class C does not imply C-complex for the class C; Berman [9] has shown that any set $\leq_m^P$-complete for a deterministic time class contains infinite polynomial time recognizable subsets, thus showing that they are not a.e. complex. The following theorem shows that sets that are very difficult to recognize are not necessarily $\leq_C$-hard.

Theorem 4.24 *There exists a recursive set A such that every infinite subset of A is a* TIME(2^{poly})*-core, and A is not $\leq_C$-hard for* TIME(2^{poly}).

Proof. Let $A \subseteq EG_o$ be a TIME$(2^{poly})|\ EG_o$-complex set. Note that by Proposition 3.14, every infinite subset of A is a TIME(2^{poly})-complexity core for A. For all but finitely many $x \in A \subseteq EG_o$ and any polynomial p, a deterministic Turing recognizer of A requires more than $|x|^{2^{p(|x|)}}$ steps to determine membership in A. Specifically, there exists a Turing machine M_A that recognizes A and a polynomial q such that for all but finitely many $x \in EG_o$, $T_{M_A}(x) > |x|^{2^{q(|x|)}}$, and $T_{M_A}(x) = q(|x|)$ for all $x \notin EG_o$.

Now let B be a DTIME(2^n)-complex set in DTIME(2^{2n}) and assume A is $\leq_C$-hard for TIME(2^{poly}). Then there must exist a polynomial p and a Turing machine recognizor M_B of B such that

$$\forall^\infty x \in \Sigma^*(T_{M_B}(x) \leq p(|x|) \max\{T_{M_A}(y) : |y| \leq p(|x|)\}).$$

But then, for sufficiently large n,

$$\forall^\infty x \in EG_e(T_{M_B}(x) \leq p(x)q(p(|x|))),$$

and since B was assumed to be DTIME(2^n)-complex, we arrive at a contradiction. Therefore, A cannot be $\leq_C$-hard for TIME(2^{poly}). □

Therfore the existence of "hard" complexity cores alone is an insufficient criterion for $\leq_C$-hardness. The distribution of the hard instances must also be taken into consideration. We investigate this phenomena further.

Definition 4.25 *Given a Turing machine M and a function f on the natural numbers, the set of f-hard instances for M is*

$$H(M,f) = \{x \in \Sigma^* : T_M(x) > f(|x|)\}.$$

Definition 4.26 *Given a function g over the natural numbers, a set A is g-distributed if*

$$\forall^\infty x \in A \, \exists y \in A \, (|x| < |y| \leq g(|x|)).$$

Proposition 4.27 *If a recursive set A is C-complex, then for every Turing machine M_A that recognizes A and every $f \in \mathcal{F}_C$, $H(M_A, f)$ is p-distributed for any polynomial p.*

Proof. If A is C-complex, then almost every string x is an f-hard instance for $f \in \mathcal{F}_C$.□

Theorem 4.28 *If there exists a polynomial p such that for every Turing machine recognizer M_A of the set A, $H(M_A, 2^n)$ is p-distributed, then A is $\leq_C$-hard for TIME(2^{poly}).*

Proof. Let A be a set and p a polynomial such that for every Turing machine recognizer M_A of A, $H(M_A, 2^n)$ is p-distributed. Further, let B be a set in TIME(2^{poly}) and M_B a Turing recognizer of B, $T_{M_B}(x) \leq 2^{q(|x|)}$ for some polynomial q. Define

$$\hat{p}(n) = p^{i+1}(n) = \overbrace{p \circ \cdots \circ p}^{i+1 \text{ times}}(n),$$

where i is the least integer i such that $p^i(n) > q(n)$ for all sufficiently large n.

For any M_A, $\max\{T_{M_A}(y) : |y| \leq \hat{p}(n)\} \geq 2^{p_i(n)}$, since by the p-distribution of $H(M_A, 2^n)$ there must exist a y, $p^i(n) < |y| \leq p(p^i(n))$, such that $T_{M_A}(y) > 2^{|y|}$. It then follows that for $\hat{p}$ and every M_A,

$$T_{M_B}(x) \leq 2^{q(|x|)} < 2^{p^i(|x|)} \leq \max\{T_{M_A}(y) : |y| \leq \hat{p}(|x|)\}$$

for all but finitely many x, and we conclude that $B \leq_C A$.□

Corollary 4.29 *There exist $\leq_C$-complete tally sets in* E.

Proof. Let $A \subseteq \{1\}^*$ be a DTIME(2^n) $\mid 1^*$-complex set in E; Theorem 3.16 ensures the existence of such sets. For every Turing recognizer M_A of A, $T_{M_A}(1^n) > 2^n$ for all but finitely many n, hence $H(M_A, 2^n)$ is p-distributed for $p(n) = n$. □

Corollary 4.30 *There exist tally sets in* $\mathrm{TIME}(2^{poly})$ *that are* $\leq_C$*-complete.*

Corollary 4.31 *There exist tally sets in* $\mathrm{DTIME}(n^2 2^n)$ *that are* $\leq_C$*-hard for* NP.

Proof. The set constructed for Corollary 4.29 is in $\mathrm{DTIME}(n^2 2^n)$, and this set is $\leq_C$-hard for $\mathrm{TIME}(2^{poly})$. □.

A function g is a generator for $\mathrm{TIME}(2^{poly})$ if for every polynomial q there exists a polynomial p such that $g(p(n)) \geq 2^{q(n)}$. For example, $g(n) = 2^n$ is a generator for $\mathrm{TIME}(2^{poly})$.

Theorem 4.32 *A set* A *is* $\leq_C$*-hard for* $\mathrm{TIME}(2^{poly})$ *if and only if there exists a polynomial* p *such that for every Turing machine recognizer* M *of the set* A *and some generator* f *of* $\mathrm{TIME}(2^{poly})$, $H(M, f)$ *is* p*-distributed.*

Proof. ($\Leftarrow$) Let f be a generator of $\mathrm{TIME}(2^{poly})$ and $H(M_A, f)$ be p-distributed. For a set B in $\mathrm{TIME}(2^{poly})$ and M_B, $T_{M_B}(x) \leq 2^{q(|x|)}$ for some polynomial q, define $\hat{p}(n) = p^{i+1}(n)$, where i is the least integer such that $f(p^i(n)) > 2^{q(n)}$ for sufficiently large n. Such an i exists since f is a generator of $\mathrm{TIME}(2^{poly})$. The rest of the proof proceeds as in Theorem 4.28.

($\Rightarrow$) Assume false, then (at least) one of two cases must hold. We handle each case separately.

(*Case 1*) For some Turing machine M_A that accepts A, there does not exist an infinite set of f-hard instances for any generator f of $\mathrm{TIME}(2^{poly})$. Let B be a $\mathrm{DTIME}(2^n)$-complex set in $\mathrm{TIME}(2^{poly})$; $B \leq_C A$. Therefore, for every Turing machine M that recognizes A there exists a Turing machine recognizer M_B of B and a polynomial p such that all but finitely many x

$$2^{|x|} < T_{M_B} \leq p(|x|) \max\{T_M(y) : |y| \leq p(|x|)\}.$$

Specifically, for M_A and sufficiently large x

$$2^{|x|} < \max\{T_{M_A}(y) : |y| \leq p(|x|)\}.$$

But this implies $T_{M_A}(y) > 2^{|x|}$ for infinitely many y, and we arrive at a contradiction.

(*Case 2*) For every generator f of TIME(2^{poly}) and some machine M_A that accepts A, there does not exist a polynomial p such that $H(M_A, f)$ is p-distributed. Again letting B be a DTIME(2^n)-complex set in TIME(2^{poly}) implies the existence of a polynomial p such that for every Turing machine M that recognizes A

$$\forall^{\infty} x \in \Sigma^* \max\{T_M(y) : |y| \leq p(|x|)\} > 2^{|x|}.$$

Specifically for M_A, this implies

$$\forall^{\infty} x \in \Sigma^* \exists y((|x| \leq |y| \leq p(|x|)) \wedge T_{M_A}(y) > 2^{|x|}.$$

But this is merely a statement that $H(M_A, T_{M_A})$ is p-distributed, and T_{M_A} is a generator for TIME(2^{poly}). Again we arrive at a contradiction.□

Corollary 4.33 *A set A is $\leq_C$-hard for* E *if and only if there exists a polynomial p such that for every Turing machine recognizer M of the set A and some generator f of* E, *$H(M, f)$ is p-distributed.*

Therefore since every $\leq_T^P$-complete set for E is $\leq_C$-complete, it follows that "hard instances" for these sets must be at most a polynomial "distance" apart.

5 RELATIVIZATIONS OF SPECIAL CASES OF THE P $\neq$ NP QUESTION

5.1 Introduction

The remainder of this dissertation is devoted to relativization results. The study of relativized complexity classes began with the work of Baker, Gill, and Solovay [6]. They formulated questions similar to the P $\neq$ NP question for mathematical models of computers that computed with the aid of oracles. The resulting machine classes are quite similar to the class of Turing machines without an oracle, but they showed that with this slight altering of the machine model different answers to relativized questions were possible. In particular, the relativized P $\neq$ NP question has an affirmative answer for some oracles but a negative answer for other oracles. This suggests that resolving the original question requires careful analysis of the computational power of machines.

We begin this study by looking at polynomial subclasses of P and NP. Specifically, we look at relativized DTIME($O(n^i)$) $\neq$ NTIME($O(n^i)$) questions and the consequences of these results. The proofs of these results are straightforward applications of the techniques developed by Baker, *et al.* We present them in some detail, for in the following chapter we will study relativized questions of a far deeper nature, and while the techniques employed for these results are the same, the details are often obscured by technical considerations.

Book [11] showed that P = NP if and only if NTIME(n) $\subseteq$ P. A consequence of this is that if DTIME($O(n)$) = NTIME(n), then P = NP. Of course, it may be true that P = NP and DTIME($O(n)$) $\neq$ NTIME(n). Rosenberg [58] showed that DTIME(n) $\neq$ DTIME($2n$) while Book and Greibach [12], on the other hand, showed that NTIME(n) = NTIME($O(n)$). An obvious corollary of this is that DTIME(n) $\neq$ NTIME(n). Paul, Pippenger, Szemerédi and Trotter [53] have shown

that DTIME($O(n)$) $\neq$ NTIME(n). Therefore, this result is a special case of the unproven assertion that P $\neq$ NP. In this chapter we look at the question of whether such special cases can be of any help in proving P $\neq$ NP.

Obvious modification of the results of Baker, Gill and Solovay yields a recursive oracle A such that DTIME$(O(n))^A \neq$ NTIME$(n)^A$ and $\mathrm{P}^A \neq \mathrm{NP}^A$. We prove the existence of a recursive oracle B such that DTIME$(O(n))^B \neq$ NTIME$(n)^B$ and $\mathrm{P}^B = \mathrm{NP}^B$. Our conclusion is that techniques for separating complexity classes which relativize, *e.g.*, translation results, will not be sufficient for obtaining P $\neq$ NP from the now known separation of linear time classes. Moreover, we draw the same conclusion for even more general special cases of the P $\neq$ NP question, for we show that for every $i > 1$ there exists a recursive oracle B such that for every $j < i$, DTIME$(O(n^j))^B \neq$ NTIME$(O(n^j))^B$ and NTIME$(O(n))^B \subseteq$ DTIME$(O(n^i))^B$ and therefore $\mathrm{P}^B = \mathrm{NP}^B$.

5.2 Main Results

Does the fact that DTIME($O(n)$) $\neq$ NTIME(n) lead us to a proof that P $\neq$ NP? We construct oracles A and B such that

(i) DTIME$(O(n))^A \neq$ NTIME$(n)^A$ and $\mathrm{P}^A \neq \mathrm{NP}^A$.

(ii) DTIME$(O(n))^B \neq$ NTIME$(n)^B$ and $\mathrm{P}^B = \mathrm{NP}^B$.

The first task is easy to satisfy.

Theorem 5.1 *There is a recursive oracle A such that* DTIME$(O(n))^A \neq$ NTIME$(n)^A$ *and* $\mathrm{P}^A \neq \mathrm{NP}^A$.

Proof. The proof follows directly from the proof of Baker, Gill and Solovay that there exists an oracle A such that $\mathrm{P}^A \neq \mathrm{NP}^A$, and the observation that relativized NP-complete sets exist in NTIME$(n)^A$.□

Definition 5.2 *For any oracle X,*

$K(X) = \{0^i 1 0^n 1 x :$ some computation of NP_i^X accepts x in fewer than n steps.$\}$.

We will say that a string of the form 0^i10^n1x is *valid* (with respect to an oracle X) if NP_i accepts x, with X as the oracle, in less than n steps. Baker, Gill, and Solovay showed that for any oracle X, $K(X)$ is $\leq_m^P$-complete for NP^X, and therefore $\mathrm{P}^X = \mathrm{NP}^X$ if and only if $K(X) \in \mathrm{P}^X$.

Definition 5.3 *For every integer $k > 0$ and oracle X, let*

$$L_k(X) = \{x : \text{ there is a } y \in X \text{ such that } |y| = |x|^k \text{ and } y \in 1\Sigma^*\}.$$

Clearly for any oracle X, $L_k(X) \in \mathrm{NTIME}(n^k)^X$; a nondeterministic n^k time-bounded oracle Turing machine on input x nondeterministically writes a string y of length $|x|^k$ on its query tape beginning with a "1." It then accepts x if y belongs to X.

Theorem 5.4 *For every integer i there exists an oracle B such that* $\mathrm{NTIME}(n)^B \not\subseteq \mathrm{DTIME}(O(n^{i-1}))^B$ *and* $\mathrm{NTIME}(n)^B \subseteq \mathrm{DTIME}(O(n^i))^B$.

Proof. Let $\{M_j\}_{j\in\mathrm{N}}$ be an effective enumeration of deterministic $O(n^{i-1})$ time-bounded oracle Turing machines. We assume that $p_j(n)$ is a strict upper bound on the length of any computation of machine M_j. We will build the oracle set B inductively on the length of the strings in the set. During the construction of B we will have two different conditions to meet. First, we ensure that $K(B)\leq_m^P B$. Secondly, each machine in the enumeration $\{M_j\}_{j\in\mathrm{N}}$ is run with oracle B in order to diagonalize out of the class $\mathrm{DTIME}(O(n^{i-1}))^B$. The construction of B proceeds in stages. Initially, let $m = 0$ and $B = \emptyset$.

Stage n. For every $w \in \Sigma^{\leq n}$, look at each string of length n of the form 0^z10^y1w. If it is valid, then place the string $0^z10^y1w1^{n^i-n}$ into $B(n)$. This is a valid inductive step in the construction of B. In a computation of length less than n, no string of length greater or equal to n can be queried. To simulate NP_z^B on input w for $y < n$ steps, we need know only which elements of length less than n belong to B. Therefore B is well defined.

If at this stage $n > m$, and for the least element k not already examined in the enumeration of $O(n^{i-1})$ time-bounded deterministic oracle Turing machines, $p_k(n) < n^i < 2^{n-1}$, then run M_k with oracle $B(n)$ on input $x_k = 0^n$. Once machine M_k has

been examined it will never be examined again. If M_k accepts 0^n, then add nothing to $B(n)$. If M_k rejects 0^n, then add the least string of length n, the first bit being a "1," that M_k does not query to $B(n)$. We know such a string must exist since M_k can query at most $p_k(n) < 2^{n-1}$ strings. Again, this step is valid since every query M_k makes to the oracle is of length less than n^i, and for all such strings membership to B has already been decided.

Now set $m = 2^n$, thus ensuring that no string is added later to B that may affect the running of M_k on strings of length n, and go to the next stage.

Claim. $\mathrm{NTIME}(n)^B \subseteq \mathrm{DTIME}(O(n^i))^B$.

Proof of claim. Let $L \in \mathrm{NTIME}(n)^B$, therefore $L = L(NP_z^B)$ for some nondeterministic oracle Turing machine NP_z with time bound $p_z(n) = O(n)$. The function $f_z(x) = 0^z 1 0^{p_z(|x|)} 1 x$ is computable in time $O(|x|)$. So, $x \in L$ if and only if NP_z^B accepts x in $p_z(|x|)$ steps if and only if $y 1^{|y|^i - |y|} = f(y) \in B$, where $y = 0^z 1 0^{p_z(|x|)} 1 x$. But, $f(y) = f(f_z(x))$ is computable in time $\mathrm{DTIME}(O(|x|^i))$, and hence $L \in \mathrm{DTIME}(O(n^i))^B$.

Claim. $\mathrm{NTIME}(n)^B \not\subseteq \mathrm{DTIME}(O(n^{i-1}))^B$.

Proof of claim. $L_1(B) \in \mathrm{NTIME}(n)^B$. By construction, for each j, M_j^B rejects x_j if and only if some string of length $|x_j|$ beginning with a "1" belongs to B. That is, $x_j \notin L(M_j^B)$ if and only if $x_j \in L_1(B)$. Therefore $L_1(B)$ does not belong to $\mathrm{DTIME}(O(n^{i-1}))^B$. □

Corollary 5.5 *There is a recursive oracle B such that* $\mathrm{DTIME}(O(n))^B \neq \mathrm{NTIME}(n)^B$ *and* $\mathrm{P}^B = \mathrm{NP}^B$.

Proof. Let $i = 2$ and construct the oracle B as in Theorem 5.4. Since $K(B)$ is $\leq_m^P$-complete for NP^B, it is only necessary to show that $K(B) \in \mathrm{P}^B$. For this we only need to show that

$$0^i 1 0^j 1 x \in K(B) \Leftrightarrow 0^i 1 0^j 1 x 1^{n^2-n} \in B, \text{ where } n = |0^i 1 0^j 1 x|.$$

Let $w \in K(B)$, then w is of the form $0^i 1 0^j 1 x$. At stage $n = |w|$, w is found to be a valid encoding and $0^i 1 0^j 1 x 1^{n^2-n}$ is placed in B. Conversely, note that the only strings in B that begin with a "0" are perfect squares, and a string $w = 0^i 1 0^j 1 x 1^{n^2-n}$ is in B, $|w| = n^2$, only if $0^i 1 0^j 1 x$ is a valid encoding. That is, only if $0^i 1 0^j 1 x \in K(B)$. □

We have seen from Corollary 5.5 that there exist relativized worlds where the Paul, Pippenger, Szemerédi and Trotter result is expected and yet P = NP. Moreover, even more generalized results can be expected in these worlds.

Corollary 5.6 *For every integer $i > 1$ there exists a recursive oracle B such that for all $j < i$,* $\mathrm{DTIME}(O(n^j))^B \neq \mathrm{NTIME}(O(n^j))^B$ *and* $\mathrm{NTIME}(n)^B \subseteq \mathrm{DTIME}(O(n^i))^B$ *and therefore* $\mathrm{P}^B = \mathrm{NP}^B$.

Proof. The proof follows directly from Theorem 5.4 with the observation that $L_1(B) \notin \mathrm{DTIME}(O(n^j))^B$ for every $j < i$.□

Theorem 5.7 *For every positive integer i there exists a recursive oracle C such that* $\mathrm{DTIME}(n^i)^C \neq \mathrm{NTIME}(n^i)^C$ *and* $\mathrm{DTIME}(O(n^i))^C = \mathrm{NTIME}(O(n^i))^C$.

Proof. We assume an effective enumeration of deterministic oracle Turing machines such that $M_{\langle j,k\rangle}$ runs in time n^k. Thus for each natural number k, $\{M_{\langle j,k\rangle}\}_{j\in\mathrm{N}}$ is an effective enumeration of the set of all n^k time-bounded deterministic oracle Turing machines. The construction of the oracle C proceeds in stages. Initially let $C = \emptyset$.

Stage n. For each $x \in \Sigma^{\leq n}$, look at all strings of length n of the form 0^z10^y1x. If such a string is valid, then place it into $C(n)$. Now, focus attention on the least element $\langle \hat{\imath}, \hat{\jmath}, \rangle$ in the enumeration of deterministic oracle Turing machines that has not already been examined. If there exists an n_0 such that $n_0^{\hat{j}}, = n$, then run $M_{\langle \hat{\imath},\hat{\jmath}\rangle}$ with oracle $C(n)$ on input 0^{n_0}. If $M_{\langle \hat{\imath},\hat{\jmath}\rangle}$ accepts 0^{n_0}, then add nothing more to $C(n)$. If $M_{\langle \hat{\imath},\hat{\jmath}\rangle}$ rejects 0^{n_0}, then add the least string of length n, the first bit being a "1," that $M_{\langle \hat{\imath},\hat{\jmath}\rangle}$ does not query to $C(n)$. We now go to the next stage. The machine $M_{\langle \hat{\imath},\hat{\jmath}\rangle}$ so examined will never be examined again. If no integer n_0 can be found, then $M_{\langle \hat{\imath},\hat{\jmath}\rangle}$ remains unexamined, we add nothing more to $C(n)$, and we go to the next stage.

First note that for all natural numbers j, k, $M_{\langle j,k\rangle}$ is eventually examined. Secondly, at each stage n only strings of length n are placed into the oracle. Since all machines run prior to stage n can only query strings of length less than n, no string placed into the oracle at stage n will have any effect on previously run machines. Thus, we may take $C = \cup_{n=1}^{\infty} C(n)$.

Claim. For every positive integer i, $\mathrm{NTIME}(O(n^i))^C \subseteq \mathrm{DTIME}(O(n^i))^C$.

Proof of claim. Let $L \in \mathrm{NTIME}(O(n^i))^C$. Then $L = L(NP_z^C)$ for some nondeterministic polynomial time-bounded oracle Turing machine NP_z with time-bound

$p_z(n) = O(n^i)$. The function $f_z(x) = 0^z 10^{p_z(|x|)} 1x$ is computable in time $O(|x|^i)$. So $x \in L$ if and only if NP_z^C accepts x in $p_z(|x|)$ steps if and only if $f_z(x) = 0^z 10^{p_z(|x|)} 1x \in C$. Hence, $L \in \mathrm{DTIME}(O(n^i))^C$.

Claim. For every positive integer i, $\mathrm{NTIME}(n^i)^C \not\subseteq \mathrm{DTIME}(n^i)^C$.

Proof of claim. $L_i(C) \in \mathrm{NTIME}(n^i)^C$. By construction, for every natural number j there is a stage n such that $M_{\langle j,i\rangle}^C$ rejects 0^n if and only if some string, beginning with a "1," of length n^i belongs to C. But then $0^n \in L_i(C)$, and therefore $L_i(C) \notin \mathrm{DTIME}(n^i)^C$.□

Theorem 5.8 *There is a recursive oracle D such that for every positive integer i,* $\mathrm{DTIME}(O(n^i))^D \neq \mathrm{NTIME}(O(n^i))^D$ *and* $\mathrm{NTIME}(O(n^i))^D \subseteq \mathrm{DTIME}(O(n^{2i}))^D$.

Proof. We assume an effective enumeration of deterministic oracle Turing machines such that $M_{\langle j,k,c\rangle}$ runs in time $cn^k + c$. Thus for each natural number k, $\{M_{\langle j,k,c\rangle}\}_{j,c\in\mathbf{N}}$ is an effective enumeration of the set of all $O(n^k)$ time-bounded deterministic oracle Turing machines. The construction of D proceeds in stages. Initially $m = 0$ and $D = \emptyset$.

Stage n. For every $x \in \Sigma^{\leq n}$, look at each string of length n of the form $0^z 10^y 1x$. If such a string is valid, then place $0^z 10^y 1x 1^{n^2-n}$ into $D(n)$. If at this stage $n > m$, then focus attention on the least machine $M_{\langle j,k,c\rangle}$ not yet examined in our enumeration. If there exists an n_0 such that $n_0^k = n$ and $cn + c < n^2 < 2^{n-1}$, then run $M_{\langle j,k,c\rangle}$ with oracle $D(n)$ on input 0^{n_0}. If $M_{\langle j,k,c\rangle}$ accepts 0^{n_0}, then add nothing more to $D(n)$. If $M_{\langle j,k,c\rangle}$ rejects 0^{n_0}, then add the least string of length n, the first bit being a "1," that $M_{\langle j,k,c\rangle}$ does not query to $D(n)$. Set $m = 2^n$ and go the next stage. If no suitable n_0 can be found, then $M_{\langle j,k,c\rangle}$ remains unexamined, we add nothing more to $D(n)$, m remains unchanged, and we go to the next stage.

The details of the remainder of the proof are handled in a fashion similar to the previous theorems and are omitted.□

We close this chapter with one final result. Baker, Gill, and Solovay constructed an oracle E such that $\mathrm{P}^E = \mathrm{NP}^E$. This oracle results in a collapsing of the linear time-bounded classes. We present the proof of this result here for the sake of completeness.

Theorem 5.9 *There is an oracle E such that* $\mathrm{DTIME}(O(n))^E = \mathrm{NTIME}(n)^E$ *and* $\mathrm{P}^E = \mathrm{NP}^E$.

Proof. We construct the oracle E sucht that $E = K(E)$. Again this is done in stages with E initially empty. At each stage n we place all valid strings of the form 0^i10^j1x into $E(n)$. $E = \cup_{n=1}^{\infty} E(n)$, and $E = K(E)$ by definition.

Consider some $L \in \text{NTIME}(n)^E$. We want to show that L is reducible to E in linear time. L is accepted by some nondeterministic oracle Turing machine NP_i in time bounded by $p_i(n) = O(n)$. The function $f_i(x) = 0^i10^{p_i(n)}1x$ is computable in time $O(|x|)$. So, $x \in L$ if and only if NP_i^E accepts x within $p_i(|x|)$ steps if and only if $f(x) \in K(E)$. But, $K(E) = E$, $E \in \text{DTIME}(O(n))^E$, and $\text{NTIME}(n)^E \subseteq \text{DTIME}(O(n))^E$.$\square$

6 RELATIVIZATIONS OF UNAMBIGUOUS AND RANDOM POLYNOMIAL TIME CLASSES

6.1 Introduction

Valiant [67] introduced the notion of an *unambiguous* Turing machine — a nondeterministic Turing machine that has at most one accepting computation for any input. Let UP $\subseteq$ NP be the collection of languages accepted by unambiguous Turing machines in polynomial time. UP^X is the relativization of this class with respect to some oracle X. Rackoff [56] showed that there is a recursive oracle A such that $\mathrm{P}^A \neq \mathrm{NP}^A = \mathrm{UP}^A$ and there is a recursive oracle B such that $\mathrm{P}^B = \mathrm{UP}^B \neq \mathrm{NP}^B$. A natural question that arises is: Does there exist an oracle C such that $\mathrm{P}^C \neq \mathrm{UP}^C \neq \mathrm{NP}^C$? We answer this in the affirmative.

The proof of the result for UP involves a combinatorial argument for which we have developed a pebbling game. This technique is of interest in itself, and a natural generalization of this game is used in solving an open problem of Book, Long and Selman [13].

The question of whether UP = NP is closely related to the question of whether there exist NP-hard public-key cryptosystems (PKCS). Even, Selman, and Yacobi [22] have shown that if promise problems associated with such systems do not exist then UP $\neq$ NP. Since the promise problem for a PKCS and sets in UP are very similar, and any algorithm solving the cracking problem of the PKCS should need more than polynomial time for all sufficiently large codes, we in addition want P^C-immune sets to exist in UP^C.

The class RP $\subseteq$ NP, the common class of problems having efficient randomized algorithms, was defined by Adleman and Manders [1]. A set A belongs to RP if and only if there exists a nondeterministic polynomial-time bounded Turing machine M

such that $A = L(M)$ and for each $x \in A$, M accepts x with probability at least $1/2$. Rackoff showed, analogous to the results for UP, that there is an oracle D such that $\mathrm{P}^D \neq \mathrm{RP}^D = \mathrm{NP}^D$, and there is an oracle E such that $\mathrm{P}^E = \mathrm{RP}^E \neq \mathrm{NP}^E$. Again the question arises: Is there an oracle F such that $\mathrm{P}^F \neq \mathrm{RP}^F \neq \mathrm{NP}^F$? Such an oracle is provided by Sipser's construction [65] of a recursive X such that RP^X contains no complete set.

A secure cryptosystem should not be susceptible to cryptanalytic attack by efficient randomized algorithims. It should not even be "crackable" by an efficient randomized algorithm infinitely often. Hence, we would like to know whether a language in UP exists which is RP-immune. In fact, we will show that both inequalities in $\mathrm{P}^F \neq \mathrm{RP}^F \neq \mathrm{UP}^F$ are strong for some oracle F. Therefore, for this oracle $\mathrm{P}^F \neq \mathrm{RP}^F \neq \mathrm{NP}^F$ and both inequalities are strong.

Rackoff's results, and ours, taken together, indicate that it will be hard to prove $\mathrm{P} \neq \mathrm{UP} \neq \mathrm{NP}$ and $\mathrm{P} \neq \mathrm{RP} \neq \mathrm{NP}$. Intuitively, one believes $\mathrm{P} \neq \mathrm{UP} \neq \mathrm{NP}$ and $\mathrm{P} \neq \mathrm{RP} \neq \mathrm{NP}$. Since existence of an oracle X such that $\mathrm{P}^X \neq \mathrm{UP}^X \neq \mathrm{NP}^X$ is a necessary condition for $\mathrm{P} \neq \mathrm{UP} \neq \mathrm{NP}$, the separation results obtained here support our intuition about the non-relativized world. The same can be said for the $\mathrm{P} \neq \mathrm{RP} \neq \mathrm{NP}$ question.

6.2 Main Results

Let $\mathrm{UP} \subseteq \mathrm{NP}$ be the collection of languages accepted by unambiguous Turing machines in polynomial time. A characterization of UP is that L belongs to UP if and only if there is some polynomial-time computable predicate $P(x, y)$ and constant k such that $L = \{x : \text{there exists a } y \text{ such that } |y| = |x|^k \text{ and } P(x,y)\} = \{x : \text{there is a unique } y \text{ such that } |y| = |x|^k \text{ and } P(x,y)\}$. $\mathrm{RP} \subseteq \mathrm{NP}$ is the collection of languages L such that for some polynomial time-computable predicate $P(x, y)$ and a constant k, $L = \{x : \text{there exists a } y \text{ such that } |y| = |x|^k \text{ and } P(x,y)\} = \{x : \text{there exist at least } 2^{|x|^k - 1} \text{ values of } y \text{ such that } |y| = |x|^k \text{ and } P(x,y)\}$.

We say that a nondeterministic Turing machine M is *unambiguous on* $\Sigma^{\leq n}$ if on each accepted input of length n or less, M has a unique accepting path. Conversely, a nondeterministic Turing machine M is *ambiguous on* $\Sigma^{\leq n}$ if on some accepted input

of length n or less, M has more than one accepting path.

Before we state and prove our theorems we must first introduce some terminology and state a combinatorial lemma. A *board* is an $m \times m$ matrix over $\{0,1\}$. A *square* of the board is an element, b_{ij}, of the matrix, where $i,j \leq m$. There are m^2 squares for every $m \times m$ board.

We describe a very simple pebbling game. Given a pebble, we may place it on any square of the board that does not already have a pebble on it. We denote a pebble on a square by a "1." The board is *covered* if the following two conditions are met for all $i \leq m$ and all $j \leq m$:

(1) $b_{ii} = 1$.

(2) $b_{ij} = 0 \rightarrow b_{ji} = 1$.

The object of the game is to cover the board.

Lemma 6.1 *At least $\lceil m^2/2 \rceil$ pebbles are necessary to cover an $m \times m$ board.*

The proof is trivial.

Definition 6.2 *For any oracle X,*

$$\begin{aligned} L_0(X) &= \{0^n : \text{there exists a unique } y \in X \text{ such that } |y| = n \text{ and} \\ &\qquad n \text{ is odd}\}, \\ L_1(X) &= \{0^n : \text{there is a } y \in X \text{ such that } |y| = n \text{ and } n \text{ is even}\}. \end{aligned}$$

Theorem 6.3 *There is a recursive oracle A such that $\mathrm{P}^A \neq \mathrm{UP}^A \neq \mathrm{NP}^A$.*

Proof. For every oracle X, $\mathrm{P}^X \subseteq \mathrm{UP}^X \subseteq \mathrm{NP}^X$. Therefore, it is sufficient to construct an oracle A containing at most one string of length n, for each odd n, such that $L_0(A) \notin \mathrm{P}^A$ and $L_1(A) \notin \mathrm{UP}^A$. (Note that $L_0(A) \in \mathrm{UP}^A$ and $L_1(A) \in \mathrm{NP}^A$.) We build A in stages. Initially $m = 0$ and $A = \emptyset$.

Stage i. If i is odd, we look at the least element j not already examined in the enumeration of polynomial time-bounded deterministic oracle Turing machines. Once a machine P_j has been examined it will never be examined again. Machine P_j has a polynomial bound $p_j(n)$. Pick an odd integer n, $n > m$, so large that $p_j(n) < 2^n$. Run P_j with oracle $A(i)$ on input $x_j = 0^n$. If $P_j^{A(i)}$ accepts 0^n, add nothing to $A(i)$

at this stage. Otherwise, if $P_j^{A(i)}$ rejects 0^n, add to A the least string of length n not queried during the computation of $P_j^{A(i)}$ on input 0^n. (We know such a string exists.) We have thus added a single string of odd length to A. Finally, set $m = 2^n$ and go to the next stage.

If i is even we look at the least element k not already examined in the enumeration of polynomial time-bounded nondeterministic oracle Turing machines. Again, once machine NP_k is examined it will never be examined again. Machine NP_k has a polynomial bound $p_k(n)$. Pick an even integer n, $n > m$, so large that $p_k(n) < 2^{n-1}$. If $NP_k^{A(i)}$ is ambiguous on $\Sigma^{\leq n}$ we add nothing to $A(i)$ at this stage, set $m = 2^n$, and go to the next stage.

If $NP_k^{A(i)}$ is unambiguous on $\Sigma^{\leq n}$, run $NP_k^{A(i)}$ on input $x_k = 0^n$. If $NP_k^{A(i)}$ accepts 0^n, add nothing to $A(i)$ at this stage. If $NP_k^{A(i)}$ rejects 0^n then add one or more strings of length n to $A(i)$ such that NP_k either still rejects 0^n, or accepts 0^n ambiguously. We will show that such strings can always be found. Now we set $m = 2^n$ and go to the next stage.

Claim. A nonempty subset $X \subseteq \Sigma^n$ exists such that $NP_k^{A(i)\cup X}$ either rejects 0^n or accepts 0^n ambiguously.

Proof of claim. We can, by exhaustive search, determine whether there exists a nonempty $X \subseteq \Sigma^n$ such that $NP_k^{A(i)\cup X}$ rejects 0^n. If we find such an X, then we are finished.

If such a subset cannot be found, then it must be the case that for all nonempty subsets X of Σ^n, $NP_k^{A(i)\cup X}$ accepts 0^n. In particular, $NP_k^{A(i)}$ does not accept 0^n, but for each string $x \in \Sigma^n$, $NP_k^{A(i)\cup\{x\}}$ does accept 0^n. An accepting path of $NP_k^{A(i)\cup\{x\}}$ on input 0^n is called a *critical path for* x. Note that every string x of length n has a critical path. If for some $x \in \Sigma^n$ there exists more than one critical path, then $NP_k^{A(i)\cup\{x\}}$ is ambiguous on 0^n. In this case take $X = \{x\}$ to settle our claim. Hence, we assume that there exists exactly one critical path for each string x. Therefore, there exist at most 2^n critical paths.

Denote $cr(x)$ as the set of queries of length n made in the critical path for x. Clearly $x \in cr(x)$. Any change in an answer to a query in $cr(x)$ may affect the resulting computation. If we place a string $y \in \Sigma^n$ into the oracle and $y \notin cr(x)$, then the addition of y to the oracle is oblivious on the critical path for x. Therefore, if we

can find strings $x, y \in \Sigma^n$ such that $x \notin cr(y)$ and $y \notin cr(x)$, then by placing both x and y into $A(i)$ there will exist two distinct computation paths that accept 0^n. We show that such strings can be found by reducing this problem to the board covering game described earlier.

Fix an ordering, $x_1, x_2, x_3, \cdots$, of the strings of Σ^n. The success, or failure, in finding strings $x_i \notin cr(x_j)$ and $x_j \notin cr(x_i)$ is equivalent to determining whether a $2^n \times 2^n$ board can be covered with a given number of pebbles. The "pebbles" in this game are the queries of length n made in each critical path of a string in Σ^n, *i.e.*, $b_{ij} = 1 \Leftrightarrow x_j \in cr(x_i)$. If the board is covered then for each $x_j \notin cr(x_i)$ ($b_{ij} = 0$) we have $x_i \in cr(x_j)$ ($b_{ji} = 1$). If the board is not covered, either for some i, $x_i \notin cr(x_i)$ ($b_{ii} = 0$), which can never happen, or for some i and j, $x_j \notin cr(x_i)$ and $x_i \notin cr(x_j)$ ($b_{ij} = 0$ and $b_{ji} = 0$). If this is the case, then we have found suitable strings.

Each critical path is of depth at most $p_k(n)$. Since there exist at most 2^n critical paths, there are at most $p_k(n)2^n$ pebbles, and since

$$p_k(n)2^n < 2^{n-1}2^n = 2^{2n-1},$$

it follows from Lemma 6.1 that we cannot cover the board. Therefore, there must be strings x_i and x_j such that $x_j \notin cr(x_i)$ and $x_i \notin cr(x_j)$, and this proves our claim.

To complete the proof of the theorem we need only to show $L_0(A) \notin \mathrm{P}^A$ and $L_1(A) \notin \mathrm{UP}^A$. $L_0(A) \notin \mathrm{P}^A$ by the usual argument (*cf.* [6]). Suppose $L_1(A) \in \mathrm{UP}^A$, then $L_1(A)$ is accepted by some unambiguous NP_i. At some stage k, and for some integer n, NP_i is run with oracle $A(k)$. By assumption $NP_i^{A(k)}$ is unambiguous on $\Sigma^{\leq n}$. But $0^n \in L_1(A)$ if and only if 0^n is rejected by $NP_i^{A(k)}$ or 0^n is accepted ambiguously. Since we assumed NP_i is unambiguous, we have a contradiction; $L_1(A) \notin \mathrm{UP}^A$.□

Theorem 6.4 *There exists a recursive oracle B and a language $L(B)$ such that $L(B)$ is P^B-immune and $L(B) \in \mathrm{UP}^B$.*

Proof. The construction is basically the same as the one given in the proof of the Immunity theorem in [61]. However, our theorem does not follow from the Immunity theorem, and so we give a straightforward proof here.

We build the oracle B in stages. $T(i)$ is a finite set of indices at stage i. Initially $T(i) = B(i) = \emptyset$ and $m = 0$.

Stage i. Let $T'(i+1) = T(i) \cup \{i\}$. Choose an integer $n > m$ such that $2^{n-1} \geq \sum_{j \in T'(i+1)}(n^j + j)$. Check whether there exists an index $j \in T'(i+1)$ such that $0^n \in L(P_j^{B(i)})$. If an index exists, take the smallest such index j, define $T(i+1) = T'(i+1) - \{j\}$, $m = 2^n$, add nothing to B at this stage and go to the next stage. If such an index does not exist, choose a string of length n that is not queried by any of the machines $P_j^{B(i)}$, $j \in T'(i+1)$, on input 0^n. (We have chosen a sufficiently large n so that such a string does exist.) Add this string to B, define $T(i+1) = T'(i+1)$, $m = 2^n$ and go to the next stage. Note for each n, B contains at most one string of length n.

Let $L(B) = \{0^n :$ there exists a $y \in B$ and $|y| = n\}$. Clearly $L(B) \in \mathrm{UP}^B$. We have to show that $L(B)$ is P^B-immune. First, $\|L(B)\| = \infty$. Suppose $L(B)$ were finite; then B would also be finite. Therefore, after some stage i_0, we must always have, in stage $i > i_0$, the case that $0^n \in L(P_j^B)$ for a given n and some $j \in T'(i+1)$. Therefore j is removed from $T'(i+1)$. This means that, from stage i_0 on, all $T(i)$ have a constant length. Therefore only a finite number of sets $L(P_j^B)$ (those whose indices are never removed from any $T(i)$) do not contain some element 0^n for some n. But, there are infinitely many j with $L(P_j^B) = \emptyset$, and so we have a contradiction, and $L(B)$ must be infinite.

Now we show that no infinite subset S of $L(B)$ is equal to some $L(P_j^B)$. Suppose $S = L(P_j^B)$, $S \subseteq L(B)$ and S infinite. If j is removed from $T'(i+1)$ at some stage i, then $0^n \in L(P_j^{B(i)})$ and $0^n \in L(P_j^B)$. Therefore, $0^n \in S$ and $0^n \in L(B)$. But if $0^n \in L(P_j^{B(i)})$ at stage i, we add no strings of length n to B, therefore $0^n \notin L(B)$. This is a contradiction, so j must stay in $T(k)$ for all stages $k > j$. Therefore for all but a finite number of 0^n chosen, $0^n \notin L(P_j^B)$. But in $L(B)$ we only have elements x, where $x = 0^n$ for some chosen n, and by assumption, $L(P_j^B) \subseteq L(B)$. Therefore $L(P_j^B)$ must be finite. This is also a contradiction.□

Corollary 6.5 *There exist a recursive oracle C and a language $L(C)$ such that $\mathrm{P}^C \neq \mathrm{UP}^C \neq \mathrm{NP}^C$, $L(C)$ is P^C-immune, and $L(C) \in \mathrm{UP}^C$.*

Proof. For the proof we only need to replace the odd stages in the proof of Theorem 6.3 with the corresponding stages in the proof of Theorem 6.4, *mutatis mutandis*.□

Let $\{Pr_j\}_{j\in\mathrm{N}}$ be a recursive enumeration of polynomial time-bounded oracle Turing machines that compute predicates of two variables. We define for any oracle X, each $k \in \mathrm{N}$, and for each Pr_j^X, $j \in \mathrm{N}$, the following language:

$$L(Pr_j^X, k) = \{x : \text{ there exists a } y \text{ such that } |y| = |x|^k \text{ and } Pr_j^X(x,y)\}.$$

(Pr_j^X, k) is *random* if and only if $L(Pr_j^X, k) = \{x :$ there exist $2^{|x|^{k-1}}$ distinct $y \in X$ such that $|y| = |x|^k$ and $Pr_j^X(x,y)\}$. Note that $L \in \mathrm{RP}^X$ if and only if there exists a j and k such that (Pr_j^X, k) is random and $L = L(Pr_j^X, k)$. We assume that $(|x| + |y|)^j + j$ is the time bound of Pr_j on input (x,y). Thus, on input (x,y), where $|y| = |x|^k$, Pr_j runs for at most $|x|^{kj} + j$ steps.

Theorem 6.6 *There exist a recursive oracle D and a language $L(D)$ such that $L(D) \in \mathrm{UP}^D$ and $L(D) \notin \mathrm{RP}^D$.*

Proof. Let $L(X) = \{0^n :$ there exists a $y \in X$ such that $|y| = n\}$. To show that $L(X) \notin \mathrm{RP}^X$, it is sufficient to show:

$$\forall j,k \in \mathrm{N}\ (Pr_j^X, k) \text{ random} \rightarrow L(X) \neq L(Pr_j^X, k).$$

We build the oracle D in stages. Initially $D = \emptyset$ and $m = 0$.

Stage i. We look at $(Pr_j^{D(i)}, k)$ where $\langle j,k\rangle = i$. Choose an integer $n > m$ such that $2^{n-1} > n^{kj} + j$. Determine whether $(Pr_j^{D(i)}, k)$ is random on $\Sigma^{\leq n}$. If it is not random, set $m = 2^n$, add nothing to D at this stage and go to the next stage. If it is random, and $0^n \in L(Pr_j^{D(i)}, k)$, set $m = 2^n$, add nothing to D at this stage and go to the next stage.

If $0^n \notin L(Pr_j^{D(i)}, k)$, we must find a string x, $|x| = n$, such that 0^n is not accepted by $(Pr_j^{D(i)\cup\{x\}}, k)$ with probability $\geq 1/2$. Such a string exists. Add x to the oracle, set $m = 2^n$ and go to the next stage.

Claim. If $0^n \notin L(Pr_j^{D(i)}, k)$, a string $x \in \Sigma^n$ exists such that 0^n is not accepted by $(Pr_j^{D(i)\cup\{x\}}, k)$ with probability $\geq 1/2$.

Proof of claim. For each y, $|y| = n^k$, a (deterministic) path of $Pr_j^{D(i)}$ on $(0^n, y)$ leads to a reject state. There are 2^{n^k} such paths; along each path there may be oracle queries of the form "$x \in D$?," where $|x| = n$, that are answered "no." If adding any x to the oracle causes the machine to accept 0^n with probability $\geq 1/2$, then there

are 2^{n^k-1} paths such that the machine queried the oracle about x on these paths; the corresponding changes in the responses caused the machine to accept. We say these are *critical queries* for x. For the 2^n strings of length n we have at least $2^n 2^{n^k-1}$ critical queries. The length of any path is limited by $n^{kj}+j$. The total number of possible critical queries is $(n^{kj}+j)2^{n^k} < 2^n 2^{n^k-1}$. A string x, $|x| = n$, such that 0^n is not accepted by $(Pr_j^{D(i)\cup\{x\}}, k)$ with probability $\geq 1/2$ must exist.

From the construction it follows that $L(D) \notin \mathrm{RP}^D$. $L(D) \in \mathrm{UP}^D$ since for every length n there is at most one string of length n in D.$_\square$

Corollary 6.7 *There exists a recursive oracle E such that $\mathrm{P}^E \neq \mathrm{RP}^E \neq \mathrm{UP}^E$, RP^E contains a P^E-immune set and UP^E contains an RP^E-immune set.*

Proof. Let $L_0(X)$ and $L_1(X)$ be as defined in Definition 6.2. We build E in stages. At the odd stages we "slowly diagonalize" $L_0(E)$ out of P^E. That is, we proceed as in the proof of Theorem 6.4, but instead of adding one string of length n to the oracle, add 2^{n-1} strings to the oracle. Hence, $L_0(E) \in \mathrm{RP}^E$. In the even stages, we slow down the construction in the proof of Theorem 6.6 so that $L_1(E)$ is RP^E-immune.$_\square$

Balcázar and Russo [7] contain a variety of relativization results of probabilistic complexity classes, and some of these results overlap with Corollary 6.7. They independently prove the existence of a recursive oracle E_0 such that RP^{E_0} contains a P^{E_0}-immune set, and they prove the existence of a recursive oracle E_1 such that $\mathrm{NP}^{E_1} \cap \mathrm{co}-\mathrm{RP}^{E_1}$ contains an RP^{E_1}-immune set.

The following corollary shows that each of the classes P, RP, UP and NP can be made distinct. Its proof is a simple extension of Corollary 6.7. Rather than diagonalize in two stages, we diagonalize in three stages, and in the third stage diagonalize NP out of UP via the techniques used in Theorem 6.3.

Corollary 6.8 *There exists a recursive oracle F such that $\mathrm{P}^F \neq \mathrm{RP}^F \neq \mathrm{UP}^F \neq \mathrm{NP}^F$, RP^F contains a P^F-immune set and UP^F contains an RP^F-immune set.*

We raise the following open questions:

(1) Is there an oracle X such that NP^X contains UP^X-immune sets?

(2) Are there an oracle X and a language $L(X)$ such that $L(X) \in \mathrm{RP}^X$ and $L(X) \notin \mathrm{UP}^X$?

Now we will apply our combinatorial technique to solve an open problem raised by Book, Long, and Selman [13] which studies properties of restricted forms of relativizations of NP.

Definition 6.9 *Let M be an oracle machine. For any set X and any input string x of M, let $QA(M, X, x) = \{q$: in some accepting computation of M relative to X on input x, the oracle is queried about $q\}$.*

Definition 6.10 *Let X be a set.* $\mathrm{NP.ACC.DEP}^X$ *is the class of languages L such that $L \in \mathrm{NP}^X$ is witnessed by a machine M such that for some polynomial q, and for all x, $\|QA(M, X, x)\| \leq q(|x|)$.*

Obviously for all sets X we have $\mathrm{UP}^X \subseteq \mathrm{NP.ACC.DEP}^X \subseteq \mathrm{NP}^X$.

Theorem 6.11 *There is a recursive oracle F such that* $\mathrm{P}^F \neq \mathrm{UP}^F \neq \mathrm{NP.ACC.DEP}^F$.

Theorem 6.11 follows from a simple modification of the proof of Theorem 6.3. Namely, we may ensure that in the even stages of the proof we always add at most two strings of any even length to the oracle. Hence $L_1(F) \in \mathrm{NP.ACC.DEP}^F$.

Book, Long and Selman raised and left open the question of whether there exists an oracle X such that $\mathrm{NP.ACC.DEP}^X \subset \mathrm{NP}^X$. This is so, and to prove this fact we first describe a generalization of the pebbling game defined earlier.

A *board* is a c-dimensional m-element matrix (*i.e.* m entries in each dimension) over $\{0, 1\}$, where c and m are positive integers. A *square* of the board is an element, $b_{\langle i_1, \cdots, i_c \rangle}$, of the matrix, where $i_1, \cdots, i_c \leq m$. A c-dimensional m-element board contains m^c squares. Let $I = \{\langle i_1, \cdots, i_c \rangle : i_1, \cdots, i_c \leq m\}$ be the set of all ordered c-tuples. A c-tuple is denoted by $\vec{i}$. Let $i_j \in \vec{i}$ if and only if $\vec{i} = \langle i_1, \cdots, i_j, \cdots, i_c \rangle$, *i.e.*, i_j is a component of the c-tuple $\vec{i}$. We say that $\vec{i}$ is *pairwise disjoint* if for all $i_j, i_k \in \vec{i}$, $j \neq k$ implies $i_j \neq i_k$.

The pebbling game is played as described earlier. Namely, given a pebble, denoted by a "1," we may place it on any square not already covered by a pebble. The object of the game is to cover the board with a given number of pebbles. The board is *covered* if the following conditions hold:

(1) $b_{\vec{i}} = 1$ for all $\vec{i}$ that are not pairwise disjoint.

(2) For all pairwise disjoint $\vec{i}$ there is a permutation π such that $b_{\pi(\vec{i})} = 1$.

Lemma 6.12 $\binom{m}{c} + m^c - m!/(m-c)!$ *pebbles are needed to cover a c-dimensional m-element board.*

Proof. Let $\Delta = \{\vec{i} : \vec{i} \text{ is not pairwise disjoint}\}$. If the board is covered, then for each $\vec{i} \in \Delta$, $b_{\vec{i}} = 1$. Also, if the board is covered, then for each $\vec{i} \in I - \Delta$, $b_{\pi(\vec{i})} = 1$ for some permutation π. There are $c!$ permutations for each $\vec{i}$, and $\|I - \Delta\| = m!/(m-c)!$. Therefore, the total number of pebbles needed to cover the board is

$$\frac{m!}{(m-c)!c!} + m^c - \frac{m!}{(m-c)!}$$

which proves our claim.□

Theorem 6.13 *There exists a recursive oracle G such that* $\text{NP.ACC.DEP}^G \neq \text{NP}^G$.

Proof. Let $L(G) = \{0^n : \text{there exists a } y,\ y \in G \text{ and } |y| = n\}$. G is constructed in stages such that $L(NP_i^G) \neq L(G)$ for all $L(NP_i^G) \in \text{NP.ACC.DEP}^G$. Initially $m = 0$ and $G = \emptyset$.

Stage $i = \langle j, k \rangle$. We examine $NP_j^{G(i)}$ (with polynomial time bound $p_j(n)$) and polynomial $p_k(n) = n^k + k$. Choose an integer n, $n > m$, so large that $p_k(n) < 2^{n/3} - 1$ and $p_j(n) < 2^n$. At this stage we will ensure that either $L(NP_j^G) \neq L(G)$ or $\|QA(NP_j, G, 0^n)\|$ is not bounded by $p_k(n)$. Run $NP_j^{G(i)}$ on $\Sigma^{\leq n}$. If $NP_j^{G(i)}$ accepts any string x, $|x| \leq n$, and $\|QA(NP_j, G(i), x)\| > p_k(|x|)$, then we add nothing to $G(i)$ at this stage, set $m = 2^n$, and go to the next stage.

If on each accepted string x, $|x| \leq n$, $\|QA(NP_j, G(i), x)\| \leq p_k(|x|)$, then run $NP_j^{G(i)}$ on input 0^n. If $NP_j^{G(i)}$ accepts 0^n, then we add nothing to $G(i)$, set $m = 2^n$ and go to the next stage.

If $NP_j^{G(i)}$ rejects 0^n, then we add $X \subseteq \Sigma^n$ to $G(i)$ such that either $NP_j^{G(i) \cup X}$ still rejects 0^n or $\|QA(NP_j, G(i) \cup X, 0^n)\| > p_k(n)$. We will show that such strings can always be found. Now set $m = 2^n$ and go to the next stage.

Claim. A non-empty subset $X \subseteq \Sigma^n$ exists such that $NP_j^{G(i) \cup X}$ either rejects 0^n or accepts 0^n but $\|QA(NP_j, G(i) \cup X, 0^n)\| > p_k(n)$.

Proof of claim. We can, by exhaustive search, determine whether there exists $X \subseteq \Sigma^n$ such that $NP_j^{G(i) \cup X}$ rejects 0^n. If we find such an X, then we are finished.

If such a subset cannot be found, then it must be the case that for all nonempty subsets X of Σ^n, $NP_j^{G(i)\cup X}$ accepts 0^n. In particular, $NP_k^{G(i)}$ does not accept 0^n, but for each string $x \in \Sigma^n$, $NP_k^{G(i)\cup\{x\}}$ does accept 0^n. $QA(NP_j, G(i)\cup\{x\}, 0^n)$ is the set of queries made on the accepting paths of NP_j with the oracle $G(i)\cup\{x\}$. In this context we denote this set $QA(x)$. To every string x of length n there exists a non-empty set $QA(x)$. Obviously, there exist no more than 2^n such sets. If $\|QA(x)\| > p_k(n)$ for some $x \in \Sigma^n$, then take $X = \{x\}$, and the claim is proved. Hence, we can assume without loss of generality that $\|QA(x)\| \leq p_k(n)$ for each $x \in \Sigma^n$.

Clearly $x \in QA(x)$. Any change in an answer to a query in $QA(x)$ may affect the resulting computation. If $y \notin QA(x)$ and y is placed in the oracle, then the addition of y to the oracle is oblivious on accepting computation paths of $NP_j^{G(i)\cup\{x\}}$. Therefore, if we can find a set of strings X in Σ^n such that $\|X\| = p_k(n) + 1$ and for all $x, y \in X$, $x \neq y$ implies $x \notin QA(y)$, then $\|QA(NP_j, G(i) \cup X, 0^n)\| > p_k(n)$. We show that such strings can be found by reducing this problem to the generalized board covering game.

Fix an ordering, $x_1, x_2, x_3, \cdots$, of the strings of Σ^n. Let us see that the success, or failure, in finding a suitable X is equivalent to determining whether a $(p_k(n) + 1)$-dimensional 2^n-element board can be covered with a given number of pebbles. We will pebble the board in the following way: We place a pebble on every $\vec{i} \in I$ which is not pairwise disjoint, because $x_r \in QA(x_r)$ for each $x_r \in \Sigma^n$. Now let us assume that $x_r \in QA(x_s)$, $r \neq s$. Then, for each maximal subset S of I whose elements contain r and s, are pairwise disjoint and identical up to permutation, we choose one of the elements of S, say $\vec{i}$, and set $b_{\vec{i}} = 1$.

If the board is covered, then for each $\vec{i} \in I$ there exists a permutation π such that $b_{\pi(\vec{i})} = 1$. But if $b_{\pi(\vec{i})} = 1$, then it must be the case that for some $r, s \in \vec{i}$, $x_r \in QA(x_s)$. Therefore, if the board is covered, then a subset X, as described above, does not exist. Conversly, if the board is not covered, then there exists a pairwise disjoint $\vec{i}$ such that $b_{\vec{i}} = 0$ and for all permutations, π, $b_{\pi(\vec{i})} = 0$. For such an $\vec{i}$ to exist it must be the case that for all $r, s \in \vec{i}$, $r \neq s$, we have $x_r \notin QA(x_s)$. The set $X = \{x_r : r \in \vec{i}\}$ is a suitable set.

Now we count the pebbles on the board. There are

$$(2^n)^{p_k(n)+1} - \frac{2^n!}{(2^n - p_k(n) - 1)!}$$

pebbles on the board for those elements in I that are not pairwise disjoint. Furthermore, since there are at most 2^n sets $QA(x)$, and since for every x, $\|QA(x)\| \leq p_k(n)$, it follows that there exist at most $p_k(n)2^n$ queries $x_r \in QA(x_s)$, $r \neq s$. For each such query, the number of pebbles placed on the board is the number of different subsets S of I as given above. There are

$$\binom{2^n - 2}{p_k(n) - 1}$$

such subsets. Hence there are at most

$$(2^n)^{p_k(n)+1} - 2^n!/(2^n - p_k(n) - 1)! + p_k(n)2^n \binom{2^n - 2}{p_k(n) - 1}$$

pebbles on the board.

By Lemma 6.12, it is sufficient to show that this number is smaller than

$$\binom{2^n}{p_k(n) + 1} + (2^n)^{p_k(n)+1} - \frac{2^n!}{(2^n - p_k(n) - 1)!}.$$

But this is equivalent to

$$p_k^2(n)(p_k(n) + 1) < 2^n - 1.$$

Since we have chosen $p_k(n) < 2^{n/3} - 1$, this inequality is fulfilled. So it follows that the board cannot be covered. There must be a suitable subset $X \subseteq \Sigma^n$ such that $x_r \notin QA(x_s)$ for all $x_s, x_r \in X$, $r \neq s$, and this proves the claim.

It follows from the usual argument that $L(G) \notin \mathrm{NP.ACC.DEP}^G$ and $L(G) \in \mathrm{NP}^G$.□

Corollary 6.14 *There is a recursive oracle H such that* $\mathrm{P}^H \neq \mathrm{UP}^H \neq \mathrm{NP.ACC.DEP}^H \neq \mathrm{NP}^H$, *and the first inequality is strong.*

We do not know whether Corollary 6.14 can be strengthed so that all inequalities are strong.

6.3 On the Existence of One-way Functions

Here we wish to make some remarks about the existence of one-way functions in a relativized setting. Recall that a one-way function is a $1-1$, honest function that is computable in polynomial time but whose inverse is not polynomial time computable. One-way functions are known to play a critical role in complexity issues surrounding public-key cryptosystems. It is observed in Grollmann and Selman [32] that one-way functions exist if and only if $\mathrm{P} \neq \mathrm{UP}$, so the results of Rackoff, as well as here, show that there do exist relativized worlds in which one-way functions exist.

It is also of interest to know whether there exist one-way functions with range belonging to P. Indeed, Grollmann and Selman show that this existence question is equivalent to whether $\mathrm{P} \neq \mathrm{UP} \cap \mathrm{co}-\mathrm{UP}$. Using the techniques developed in the previous section we prove the existence of a recursive oracle A such that $\mathrm{P}^A \neq \mathrm{UP}^A \cap \mathrm{co}-\mathrm{UP}^A \neq \mathrm{NP}^A$, and therefore, relative to oracle A, there exist one-way functions with easy to recognize range.

Combining results of Baker, Gill and Solovay [6] and Rackoff [56], we have the following theorem.

Theorem 6.15 *There exists a recursive oracle A such that $\mathrm{P}^A \neq \mathrm{UP}^A = \mathrm{NP}^A$ and NP^A is closed under complementation.*

Proof. Recall that for any oracle X, the language $K(X) = \{0^i 1 0^j 1 x : \text{some computation of } NP_i^X \text{ accepts } x \text{ in fewer than } j \text{ steps}\}$ is many-one complete for NP^X. Clearly, NP^X is closed under complementation if and only if $\overline{K}(X) \in \mathrm{NP}^X$, where $\overline{K}(X)$ is the complement of $K(X)$. Define $L(A) = \{0^n : \text{there exists a } y \in A \text{ such that } |y| = n\}$. It is sufficient to construct an oracle A such that:

(i) $L(A) \in \mathrm{NP}^A - \mathrm{P}^A$.

(ii) $w \in K(A)$ if and only if there exists a string v, $|v| = |w| - 1$, such that $0vw \in A$.

(iii) $w \in \overline{K}(A)$ if and only if there exists a string v, $|v| = |w| - 1$, such that $1vw \in A$.

We build the oracle A in stages. At stage n we decide the membership in A of all strings of length n. During the construction some strings are *reserved* for $\overline{A}$, *i.e.*, designated as nonmembers of A. Initially, $A = \emptyset$.

Stage $n = 2m$. For every string w of length m of the form 0^z10^y1x such that $NP_z^{A(n)}$ accepts x in less than y steps, find the least string v_0 of length m, beginning with a "0," such that v_0w is not reserved for $\overline{A}$ and place v_0w into $A(n)$. For every string w of length m of the form 0^z10^y1x such that $NP_z^{A(n)}$ does not accept x in less than y steps, find the least string v_1 of length m, beginning with a "1," such that v_1w is not reserved for $\overline{A}$ and place v_1w into $A(n)$. Go to the next stage.

Stage $n = 2m + 1$. We look at the least element j not already examined in the enumeration $\{P_i\}$ of polynomial time-bounded deterministic oracle Turing machines. Once a machine P_j has been successfully diagonalized, it will never be examined again. If any string of length $\geq n$ has been reserved for $\overline{A}$, or if $p_j(n) \geq 2^{m-1}$, then add no elements to A at this stage. Otherwise, run $P_j^{A(n)}$ on input 0^n and reserve for $\overline{A}$ all strings of length $\geq n$ queried during this computation. If P_j accepts 0^n, then add no element to A. If P_j rejects 0^n, then add to $A(n)$ the least string of length n not queried. Go to the next stage.

First, note that every machine P_j in our enumeration of polynomial time-bounded deterministic oracle Turing machines is examined at some stage, and this guarantees that $L(A) \notin \mathrm{P}^A$. At any odd stage $2m + 1$, at most $p_i(n) < 2^{m-1}$ strings are queried, so fewer than 2^{m-1} strings of length $2m$ can be reserved for $\overline{A}$ at odd stages before stage $2m$. Therefore, every string v of length m is prefixed by at least one string of equal length that begins with a "0," and one string of equal length that begins with a "1," that is never reserved for $\overline{A}$. By construction, $w \in K(A)$ if and only if there exists a unique string v such that $0vw \in A$ and $|0vw| = 2|w|$. Therefore, $K(A) \in \mathrm{UP}^A$ and $\mathrm{NP}^A = \mathrm{UP}^A$. Similarly, $w \in \overline{K}(A)$ if and only if there exists a unique string v such that $1vw \in A$, $|1vw| = 2|w|$, and $\overline{K}(A) \in \mathrm{UP}^A = \mathrm{NP}^A$. □

Corollary 6.16 *There exists a recursive oracle* A *such that* $\mathrm{P}^A \neq \mathrm{UP}^A \cap \mathrm{co} - \mathrm{UP}^A$.

While the results of Theorem 6.15, Corollary 6.16 provide us with the desired one-way functions, they do so in a relativized world that seems counterintuitive. Intuitively, one believes that $\mathrm{P} \neq \mathrm{UP} \neq \mathrm{NP}$. The results of the previous section support this intuition. Do our desired one-way functions exist in such a relativized world? The answer is provided in the following theorem.

Theorem 6.17 *There exists a recursive oracle B such that* $\mathrm{P}^B \neq \mathrm{UP}^B \cap \mathrm{co}-\mathrm{UP}^B \neq \mathrm{NP}^B$.

Proof. For any oracle X, let

$$\begin{aligned} L_0(X) &= \{x : |x| \text{ is odd and there is a string } 0y \in X \text{ with } |0y| = |x|\}, \\ L_1(X) &= \{x : |x| \text{ is even and there is a } y \in X \text{ with } |y| = |x|\}. \end{aligned}$$

We construct an oracle B such that $L_0(B) \in \mathrm{UP}^B \cap \mathrm{co}-\mathrm{UP}^B - \mathrm{P}^B$ and $L_1(B) \in \mathrm{NP}^B - \mathrm{UP}^B$. To force $L_0(B) \in \mathrm{UP}^B \cap \mathrm{co}-\mathrm{UP}^B$ we require that for every odd n there is exactly one string $0y$ of length n in B if and only if there is no string $1y$ of length n in B. Therefore, $\overline{L_0}(B) = \{x : |x| \text{ is even, or } |x| \text{ is odd and there is a string } 1y \in B \text{ with } |1y| = |x|\}$. If we require that for every odd n there is exactly one string $1y$ of length n in B if and only if there is no string $0y$ of length n in B, then $\overline{L_0}(B) \in \mathrm{UP}^B$ also. We build B in stages. Initially $m = 0$ and $B = \emptyset$.

Stage i. If i is odd we look at the least element k not already examined in the enumeration $\{P_i\}$ of polynomial time-bounded deterministic oracle Turing machines. Once a machine P_k has been successfully diagonalized, it will never be examined again. Pick an odd integer n, $n > m$, so large that $p_k(n) < 2^{n-1}$. For all odd integers q, $m \leq q < n$ and $n < q < 2^n$, place 1^q into $B(i)$. Run $P_k^{B(i)}$ on input $x_k = 0^n$. If P_k accepts 0^n, then add to B the least string of length n beginning with a "1" not queried during the computation of $P_k^{B(i)}$ on input 0^n. Otherwise, add the least string of length n beginning with a "0" not queried during the computation of $P_k^{B(i)}$ on input 0^n. We have thus added, for each odd integer q, $m \leq q < 2^n$, a single string of length q. Set $m = 2^n$ and go to the next stage.

If i is even, then we look at the least element k not already examined in the enumeration $\{NP_i\}$ of polynomial time-bounded nondeterministic oracle Turing machines. Once a machine NP_k has been successfully diagonalized, it will never be examined again. Pick an even integer n, $n > m$, so large that $p_k(n) < 2^{n-1}$. For all odd q, $m \leq q < 2^n$, place 1^q into $B(i)$. If $NP_k^{B(i)}$ is ambiguous on $\Sigma^{\leq n}$, then we add no string of length n to $B(i)$ at this stage, set $m = 2^n$, and we go to the next stage.

If $NP_k^{B(i)}$ is unambiguous on $\Sigma^{\leq n}$, then run $NP_k^{B(i)}$ on input $x_k = 0^n$. If $NP_k^{B(i)}$ accepts 0^n, then add no string of length n to $B(i)$ at this stage. If $NP_k^{B(i)}$ rejects 0^n

then add one or more strings of length n to $B(i)$ such that NP_k either still rejects 0^n, or accepts 0^n ambiguously. Theorem 6.3 shows that such strings can always be found. Now we set $m = 2^n$ and go to the next stage.

To complete the proof of the theorem we need only to show $L_0(B) \in \mathrm{UP}^B \cap \mathrm{co} - \mathrm{UP}^B - \mathrm{P}^B$ and $L_1(B) \notin \mathrm{UP}^B$. That $L_1(B) \in \mathrm{NP}^B - \mathrm{UP}^B$ follows as in the proof of Theorem 6.3. $L_0(B) \notin \mathrm{P}^B$ by the usual diagonalization argument. But also, for each odd integer q we have added to the oracle exactly one string of length q. In particular, $x \in L_0(B)$ if and only if $|x| = q$ is odd and there exists exactly one string $0y$ of length q in B if and only if there is no string $1w$ of length q in B. Hence, $L_0(B) \in \mathrm{UP}$. Similarly, $x \in \overline{L_0}(B)$ if and only if $|x| = q$ is even or if and only if q is odd and there exists exactly one string $1y$ of length q in B if and only if there is no string $0w$ of length q in B. Hence, $\overline{L_0}(B) \in \mathrm{UP}$.□

Corollary 6.18 *There exists a recursive oracle B such that $\mathrm{P}^B \neq \mathrm{UP}^B \cap \mathrm{co} - \mathrm{UP}^B \neq \mathrm{NP}^B$ and NP^B is closed under complementation.*

Proof. The proof follows along the lines of Theorem 6.17, but with an additional stage; the odd stage of Theorem 6.15. □

We close this section with the following question: Does there exist a recursive oracle C such that $\mathrm{P}^C \neq \mathrm{UP}^C = \mathrm{NP}^C$ and NP^C is not closed under complementation?

7 CONCLUSION

We have studied structural properties [1] of intractable sets in the belief that these properties need to be understood before the hard questions of wider interest, *e.g.*, $\mathrm{P} \neq \mathrm{NP}$, can be resolved. This belief can be traced back, in part, to an analogous situation in Recursive Function Theory, and the work presented here has been influenced by the success and failure of various analogues to this theory.

The study of efficient reducibilities has been a rich source of ideas, concepts and questions in computational complexity. Unfortunately, it has not provided the techniques necessary to solve these problems. Very little is known about the structure of NP, and relativization results show that such insight is not forthcoming. This has led us to view structure questions from two different perspectives. We have examined absolute structural properties of the classes E and $\mathrm{TIME}(2^{poly})$ in hope that these results will shed light on the structural aspects of NP, and we have looked at relativized structural questions about NP.

The first part of this work was motivated by a reexamination of what it means for a set to be complete for a complexity class *via* efficient reducibilities. We extended the traditional approach of using efficient reducibilities to study structural relationships between computable sets. We defined a new noneffective binary relation $\leq_C$ that in a precise way related the computational complexity of two recursive sets. The $\leq_C$ relation can be viewed as a transitive reducibility. It is the weakest mathematically meaningful notion that captures all other efficient reducibilities, and it yields new completeness and hardness notions for complexity classes.

In investigating this relation, we found that the notion of a.e. complexity played

[1] We have never formally defined exactly what a structural property is; perhaps the words of former Supreme Court Justice Potter Stewart, who when asked about pornography replied, "I may not be able to define it, but I know it when I see it," are applicable here.

an important role, and so we first looked at this phenomena with the view of constructing a.e. complex sets for various complexity classes. This led to the derivation of a deterministic time hierarchy theorem for a.e. complex sets that was as tight as for the i.o. case, and it is a significant improvement over all other known results for a.e. complex sets.

Intuitively, the a.e. complex sets must be viewed as the "hardest" sets to compute, and it is from the strong hierarchy theorem that the first important theorems on $\leq_C$-completeness and $\leq_C$-hardness are derived. For example, it was shown that if a set was $f(n)$-complex, then it was in fact $\leq_C$-hard for DTIME($f(n)$). This is a very appealing result, for it shows that $\leq_C$-hard accurately captures the notion of being computationally hard-to-compute. Moreover, we showed that there are sets that are $\leq_C$-hard for NP that are not $\leq_T^P$-hard for NP, and we showed that there are sets that must be considered complete for E that are not even $\leq_T^P$-complete for E.

Further investigation of $\leq_C$-hardness showed that a.e. complexity was too strong a concept to accurately characterize all $\leq_C$-hard sets for a given complexity class. We derived a generalized notion of a.e. complexity and derived similar, very tight, hierarchy theorems for sets that cannot be a.e. complex for syntactic reasons, but for which, intuitively, a.e. complex notions should exist. By using the techniques developed here, it was possible to show that the hard instances for complete sets for E and TIME(2^{poly}) must have a fairly normal distribution.

The second part of this work was concerned with relativization. Information on classes relativized to oracles can often lend plausability to conjectures about the nonrelativized classes — conjectures which currently defy solution. We introduced the basic concepts and techniques employed in relativization results in Chapter 5. Here we were motavated by the fact that it is now known that deterministic linear time differs from nondeterministic linear time. We investigated, by relativization, whether this result would be of any help in solving the P $\neq$ NP. We concluded that even with this result recursion theoretic techniques would be insufficent for solving the P $\neq$ NP problem.

Finally, we studied the relationships between P, NP, and the unambiguous and random time classes UP, and RP. Questions concerning these relationships are motavated by complexity issues in public-key cryptosystems. We proved that there exists

a recursive oracle A such that $P^A \neq UP^A \neq NP^A$, and such that the first inequality is strong, *i.e.*, there exists a P^A-immune set in UP^A. Further, we constructed a recursive oracle B such that UP^B contains an RP^B-immune set. As a corollary we obtained $P^B \neq \underline{RB}^B \neq NP^B$ and both inequalities are strong. By use of the techniques employed in the proof that $P^A \neq UP^A \neq NP^A$, we were also able to solve an open problem raised by Book, Long and Selman.

There are several directions in which this work can be expanded. The very strong hierarchy theorem we obtained for deterministic time is as tight as the hierarchy result for the i.o. case. Translation lemmas have traditionally been used to derive tighter results; this is particularly true in the nondeterministic case. Unfortunately, the padding technique that is critical to these results does not carry over for the a.e. complex case. It is an open question whether there exists a tight, very strong hierarchy theorem for nondeterministic time. Can we derive similar translation results for the a.e. case? Any technique that is developed to solve this problem would seem to offer promise for the nondeterministic time case.

Generalized Kolmogorov complexity has renewed interest in the study of sparse sets and P-printable sets. We have shown how our generalized notion of a.e. complexity and hierarchy theorem can be used to answer questions in this area. Can generalized Kolmogorov complexity be used to answer questions pertaining to generalized a.e. complexity?

Our initial motivation for developing the $\leq_C$ relation was to study NP-complete sets: Are there $\leq_C$-complete sets in NP that are not $\leq_m^P$-complete for NP? We fell short of answering this question, but we did show that this is the case for $\leq_C$-hardness. We conjecture that $\leq_C$-complete for NP differs from $\leq_m^P$-completeness, but this question is unresolved.

8 BIBLIOGRAPHY

[1] Adleman, L., and K. Manders. "Reducibility, randomness, and intractibility". *Proc. 9th ACM Symposium on the Theory of Computing* 9(1977):151–153.

[2] Aho, A., J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Reading: Addison-Wesley, 1974.

[3] Allender, E. "The complexity of sparse sets in P". *Structure in Complexity Theory, Lecture Notes in Computer Science Vol. 223*, pp. 1-11. Berlin: Springer-Verlag, 1986.

[4] Allender, E., and R. Rubinstein. "P-printable sets". Manuscript. Rutgers University, 1986.

[5] Ambos-Spies, K. "On the relative complexity of subproblems of intractable problems". *STACS 85, Lecture Notes in Computer Science Vol. 182*, pp. 1-12. Berlin: Springer-Verlag, 1985.

[6] Baker, T., J. Gill, and R. Solovay. "Relativization of the P =? NP question". *SIAM Journal of Computing* 4(1975):431–442.

[7] Balcázar, J., and D. Russo. "Immunity and simplicity in relativizations of probabilistic complexity classes". Manuscript. Facultad de Informática U.P.C. Barcelona, Spain, 1985.

[8] Balcázar, J., and U. Schöning. "Bi-immune sets for complexity classes". *Mathematical Systems Theory* 18(1985):1–10.

[9] Berman, L. "On the structure of complete sets: almost everywhere complexity and infinitely often speedup". *Proc. 17th IEEE Symposium on the Foundations of Computer Science* 17(1976):76–80.

[10] Berman, L., and J. Hartmanis. "On isomorphisms and density of NP and other complete sets". *SIAM Journal of Computing* 1(1977):305–322.

[11] Book, R. "On languages accepted in polynomial time". *SIAM Journal of Computing* 1(1972):281–289.

[12] Book, R., and S. Greibach. "Quasi-realtime languages". *Mathematical Systems Theory* 4(1970):97–111.

[13] Book, R., T. Long, and A. Selman. "Quantitative relativizations of complexity classes". *SIAM Journal of Computing* 13(1984):461–487.

[14] Book, R., C. Wrathall, A. Selman, and D. Dobkin. "Inclusion complete tally languages and the Hartmanis-Berman conjecture". *Mathematical Systems Theory* 11(1977):1–8.

[15] Breidbart, S. "On splitting recursive sets". *Journal of Computer and System Sciences* 17(1978):56–64.

[16] Cobham, A. "The intrinsic computational difficulty of functions". *Proc. 1964 International Congress for Logic Methodology and Philosophy of Science*, pp. 24-30. Ed. Y. Bar-Hillel. Amsterdam: North Holland, 1964.

[17] Constable, R. "Hierarchy theorems for axiomatic complexity". *Computational Complexity*, pp. 36-63. Ed. R. Rustin. New York: Algorithmics Press, 1973.

[18] Cook, S. "The complexity of theorem proving procedures". *Proc. 3rd ACM Symposium on the Theory of Computing* 3(1971):151–158.

[19] Cook, S. "A hierarchy for nondeterministic time complexity". *Journal of Computer and System Sciences* 7(1973):343–353.

[20] Edmonds, J. "Paths, trees and flowers". *Canadian Journal of Mathematics* 17(1965):449–467.

[21] Even, S., T. Long, and Y. Yacobi. "A note on deterministic versus nondeterministic complexity". *Information and Control* 55(1982):117–124.

[22] Even, S., A. Selman, and Y. Yacobi. "The complexity of promise problems with applications to public-key cryptography". *Information and Control* 61(1984):159–173.

[23] Even, S., A. Selman, and Y. Yacobi. "Hard-core theorems for complexity classes". *Journal of the ACM* 32(1985):205–217.

[24] Ferrante, J., and C. Rackoff. "The computational complexity of logical theories". *Lecture Notes in Mathematics Vol. 718*. Ed. A. Dold, and B. Eckmann. Berlin: Springer-Verlag, 1979.

[25] Fischer, P., A. Meyer, and A. Rosenberg. "Real-time simulation of multihead tape units". *Journal of the ACM* 19(1972):590–607.

[26] Flajolet, P., and J. Steyaert. "Une formalisation de la d'algorithme de tri non récurrent". *Thèse de 3° cycle. Université Paris VII* (1973).

[27] Flajolet, P., and J. Steyaert. "On sets having only hard subsets". *Automata, Languages, and Programming, Lecture Notes in Computer Science Vol. 14*, pp. 446-457. Berlin: Springer-Verlag, 1974.

[28] Fürer, M. "The tight deterministic time hierarchy". *Proc. 14th ACM Symposium on the Theory of Computing* 14(1982):8–16.

[29] Geske, J., and J. Grollmann. "Relativizations of unambiguous and random polynomial time classes". *SIAM Journal of Computing* 15(1986):511–519.

[30] Geske, J., and D. Huynh. *Hierarchies of almost everywhere complex sets.* Technical Report 86-05 Dept. Computer Science, Iowa State University Ames, Iowa 1986.

[31] Gill, J., and M. Blum. "On almost everywhere complex recursive functions". *Journal of the ACM* 21(1974):425–435.

[32] Grollmann, J., and A. Selman. "Complexity measures for public-key cryptosystems". *Proc. 25th IEEE Symposium on the Foundations of Computer Science* 25(1984):495–503.

[33] Hartmanis, J. "Generalized Kolmogorov complexity and the structure of feasible computations". *Proc. 24th IEEE Symposium on the Foundations of Computer Science* 24(1983):439–445.

[34] Hartmanis, J. "On sparse sets in NP - P". *Information Processing Letters* 16(1983):55–60.

[35] Hartmanis, J., P. Lewis, and R. Stearns. "Hierarchies of memory limited computations". *Proc. 6th IEEE Symposium on Switching Circuit Theory and Logical Design* 6(1965):179–180.

[36] Hartmanis, J., and R. Stearns. "On the computational complexity of algorithms". *Transactions of the American Mathematical Society* 117(1965):285–306.

[37] Hartmanis, J., and Y. Yesha. "Computation times of NP sets of different densities". *Theoretical Computer Science* 34(1984):17–32.

[38] Homer, S., and W. Maass. "Oracle dependent properties of the lattice of NP sets". Manuscript. Boston University, 1985.

[39] Hopcroft, J., and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Reading: Addison-Wesley, 1979.

[40] Jockusch, C. "Semirecursive sets and positive reducibility". *Transactions of the American Mathematical Society* 131(1968):420–436.

[41] Karp, R. "Reducibility among combinatorial problems". *Complexity of Computer Computations.* Ed. R. Miller, and J. Thatcher. New York: Plenum Press, 1972.

[42] Karp, R., and R. Lipton. "Some connections between nonuniform and uniform complexity classes". *Proc. 12th ACM Symposium on the Theory of Computing* 12(1980):302–309.

[43] Ko, K., P. Orponen, U. Schöning, and O. Watanabe. "What is a hard instance of a computational problem?". *Structure in Complexity Theory, Lecture Notes in Computer Science Vol. 223*, pp. 197-217. Berlin: Springer-Verlag, 1986.

[44] Ladner, R. "On the structure of polynomial time reducibility". *Journal of the ACM* 22(1975):155–171.

[45] Ladner, R., N. Lynch, and A. Selman. "A comparison of polynomial time reducibilities". *Theoretical Computer Science* 1(1975):103–123.

[46] Landweber, L., and E. Robertson. "Recursive properties of abstract complexity classes". *Journal of the ACM* 19(1972):296–308.

[47] Lynch, N. "On reducibility to complex or sparse sets". *Journal of the ACM* 22(1975):341–345.

[48] Machtey, M., and P. Young. *An Introduction to the General Theory of Algorithms.* Amsterdam: North Holland, 1978.

[49] Meyer, A., and E. McCreight. "Computationally complex and pseudo-random zero-one valued functions". *Theory of Machines and Computations*, pp. 19-42. Ed. Z. Kohavi, and A. Paz. New York: Academic Press, 1971.

[50] Meyer, A., and L. Stockmeyer. "The equivalence problem for regular expressions with squaring requires exponential space". *Proc. 13th IEEE Symposium on Switching and Automata Theory* 13(1972):125–129.

[51] Orponen, P., and U. Schöning. "The structure of polynomial complexity cores". *Automata, Languages, and Programming, Lecture Notes in Computer Science Vol. 176*, pp. 452-458. Berlin: Springer-Verlag, 1984.

[52] Paul, W. "On time hierarchies". *Proc. 9th ACM Symposium on the Theory of Computing* 9(1977):218–222.

[53] Paul, W., N. Pippenger, E. Szemerédi, and W. Trotter. "On determinism versus non-determinism and related problems". *Proc. 24th ACM Symposium on the Theory of Computing* 24(1983):429–438.

[54] Post, E. "Recursively enumerable sets of positive integers and their decision problems". *Bulletin of the American Mathematical Society* 50(1944):284–316.

[55] Rabin, M. *Degree of difficulty of computing a function and a partial ordering of recursive sets.* Technical Report 2 Hebrew University Jerusalem, Israel 1960.

[56] Rackoff, C. "Relativized questions involving probabilistic algorithms". *Journal of the ACM* 29(1982):261–268.

[57] Rogers, H. *Theory of Recursive Functions and Effective Computability.* New York: McGraw-Hill, 1967.

[58] Rosenberg, A. "Real time definable languages". *Journal of the ACM* 14(1967):645–662.

[59] Sahni, S. "Computationally related problems". *SIAM Journal of Computing* 3(1974):262–279.

[60] Sahni, S., and T. Gonzales. "P-complete approximation problems". *Journal of the ACM* 23(1976):555–565.

[61] Schöning, U., and R. Book. "Immunity, nondeterminism, and relativization". *SIAM Journal of Computing* 13(1984):329–337.

[62] Seiferas, J., M. Fischer, and A. Meyer. "Separating nondeterministic time complexity classes". *Journal of the ACM* 25(1978):146–167.

[63] Selman, A. "Analogues of semirecursive sets and effective reducibilities to the study of NP complexity". *Information and Control* 52(1982):36–51.

[64] Selman, A. "Reductions on NP and p-selective sets". *Theoretical Computer Science* 19(1982):287–304.

[65] Sipser, M. "On relativization and the existence of complete sets". *Automata, Languages, and Programming, Lecture Notes in Computer Science Vol. 140*, pp. 523-531. Amsterdam: North Holland, 1982.

[66] Turing, A. "On computable numbers, with an application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society* 5(1936):20–23.

[67] Valiant, L. "Relative complexity of checking and evaluating". *Information Processing Letters* 5(1976):20–23.

[68] von Neumann, J. "A certain zero-sum two-person game equivalent to the optimal assignment problem". *J. Von Neumann: Collected Works, Volume VI*, pp. 44-49. Ed. A. Taub. Oxford: Pergamon Press, 1963.

[69] Webb. J. *Mechanism, Mentalism and Metamathematics.* Dordrecht: D. Reidel, 1980.

[70] Young, P. "Some structural properties of polynomial reducibilities". *Proc. 15th ACM Symposium on the Theory of Computing* 15(1983):392–401.

9 ACKNOWLEDGMENTS

As Mary-Claire van Leunen, in *A Handbook for Scholars*, has observed, "Many an advisor before yours has been thanked for understanding and guidance; many a colleague for useful contributions; many a typist for accuracy and neatness; many a spouse for patience. Not one has escaped gratitude for service above and beyond the call of duty." And yet, at the risk of sounding trite, I must extend special thanks to those people who supported me in this endeavor; I would be remiss in not doing so.

My first thanks must go to Alan Selman who introduced me to computational complexity and suggested the topic for this thesis. He fostered a theoretical environment that allowed a budding graduate student to grow intellectually, and through his cajoling and insightful comments a germ of an idea flowered into this final product. Dung Huynh also deserves a special "thank you" for serving tirelessly on my committee. It was only through his insight of the Balcázar and Schöning paper that the Hierarchy Theorem for a.e. sets was possible. I would also like to thank Joachim Grollmann for the many wonderful discussions, not all technical, we had during his one year stay at Iowa State; it was, for both of us, a most productive time.

No thanks need go to the typist — any typographical errors, unfortunately, are mine. This thesis was produced using the TEX and LATEX document preparation systems, without which the readability of this thesis would have been greatly diminished.

My wife, Barbara, deserves special accolades for her patience, understanding and sacrifice during this process. Her support and encouragement was always necessary and not always acknowledged. The final word belongs to Michael, age seven, and Matthew, age three, whose arrival into our world greatly prolonged this undertaking — a small price to pay for such wondrous progeny.