COMPUTATIONAL COMPLEXITY OF REAL FUNCTIONS AND POLYNOMIAL TIME APPROXIMATIONS

DISSERTATION

 Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Ker-I Ko, B.S., M.S.

* * * * *

The Ohio State University

1979

Reading Committee:
Professor Daniel J. Moore
Professor Harvey Friedman
Professor H. William Buttelmann
Professor Jerome Rothstein

Approved By

[Signature]

Adviser
Department of Computer and Information Science
ACKNOWLEDGMENTS

I am grateful to my advisers, Professor Dan Moore and Professor Harvey Friedman, for their suggestions of the research topics and for their continuing guidance. I am greatly indebted to Professor Moore for his careful reading of the manuscript and his constant encouragement.

I would also like to thank Professor Bill Buttelmann and Professor Jerome Rothstein for having served in my reading committee.
VITA

August 1, 1950 . . . . Born - Taipei, Taiwan, Republic of China

1972 . . . . . . . B.S. in Mathematics, National Tsinghua University, Hsinchu, Taiwan, Republic of China

1974 . . . . . . . M.S. in Mathematics, The Ohio State University, Columbus, Ohio

1976 . . . . . . . M.S. in Computer and Information Science, The Ohio State University, Columbus, Ohio

1976-1978 . . . . Teaching Associate, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio

1978-1979 . . . . Research Associate, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio

FIELDS OF STUDY

Major Field: Computer and Information Science

Studies in Computational Complexity. Professor Daniel J. Moore and Professor Harvey Friedman.

Studies in Computer Systems. Professor David Hsiao.

TABLE OF CONTENTS

ACKNOWLEDGMENTS ii

VITA iii

LIST OF SYMBOLS AND NOMENCLATURE vi

PART A Computational Complexity of Real Functions

Introduction 1

Chapter I. Computable Real Functions 5

1.1. Computable Real Numbers 5
1.2. Recursively Enumerable Real Numbers 15
1.3. Relative Recursiveness and Hierarchy of Nonrecursive Real Numbers 20
1.4. Computable Real Functions 28
1.5. Domains, Ranges, and Continuity 33

Chapter II. Polynomial Time Computable Real Functions 45

2.1. Time Complexity 45
2.2. Complexity and Modulus of Continuity 51
2.3. Roots 54
2.4. Derivatives 70
2.5. Integrals 74
2.6. Maximum Values 78

Chapter III. Conclusion 81

PART B Polynomial Time Approximation to NP-hard Problems

Introduction 88

Chapter IV. Approximation Preserving Reducibility 94

4.1. Recursive Functions as Models of Optimization Problems 95
4.2. Definition
4.3. Application

Chapter V. Probabilistic Algorithms
5.1. Probabilistic Turing Machines
5.2. A Negative Result

Chapter VI. Average Case Analysis
6.1. Introduction
6.2. P-productive Sets and P-immune Sets
6.3. Density and Complexity
6.4. Summary and Open Questions

APPENDIX Definitions of Optimization Problems

LIST OF REFERENCES

Part A
Part B
LIST OF SYMBOLS AND NOMENCLATURE

α, 86
A', 20
abstract problem level, 106-107
addition, 26
algebraically closed field, 27, 165
almost everywhere complexity, 126
analytic functions, 50, 61
analyticity, 3
AP, 100
approximation algorithms, 94
simplified view of, 125
approximation preserving reducibility, 90, 94, 95
application of, 106
definition of, 103
APT, 140
arithmetic hierarchy, 22, 96
arithmetically closed field, 7
average case analysis, 89, 91, 125
Λ(x, φ), 5
(∀y)_n, 98
B, 86
binpacking problem, 88, 94, 101, 115, 148
Blum's axioms, 45
BPP, 118
B^1_x, 5
C, 4
≤_e, 86
C[0,1], 50
C^2[0,1], 64
C^m[0,1], 65
cannonical index of a finite set, 18
Cauchy sequence representation of a real number, 6
complexity,
of differential equations, 87
of recursive real functions, 45
of recursive real numbers, 45
of sequences of real numbers, 87
of Turing machines, 45
component of an open set, 36
composite Simpson's rule, 78
calculable functionals, 74
calculable real functions, 28
calculable real numbers, 5
co-NP, 90, 93
continuity, 3, 33, 85
continuous functions, 32, 40, 81
continuous method of computing
roots of a polynomial, 66-67
correct part of an approximation
algorithm, 126
creative sets, 3
$\mathcal{C}$-reducibility, 86
cylinder, 18
density of a set, 137
derivatives, 70
deterministic algorithms, 90
distribution over $\Sigma^*$, 138
$D_n$, 4
domains of partial recursive real
functions, 29, 33
$\text{DTIME}(\mathcal{E})$, 93
$\text{DTIME}(\mathcal{T})$, 92
dyadic numbers, 4
effectively speedable sets, 127
enumeration reducibility, 104
$\varepsilon$-approximable functions, 100
$\varepsilon$-approximate, 100
$\varepsilon$-$d$-approximate, 100
$(\exists x)_{m,r}$, 117
exact cover problem, 110, 115, 148
$\text{EKP}$, 50
exponential time computable,
but not polynomial time
computable real numbers, 48
real functions, 50
real numbers, 46
sets, 92
EXPR, 46
EXPTIME, 93
$(\exists y)_n$, 98

FAP, 100
feedback vertex problem, 108, 148
formal language level, 95
fully polynomial time approximable functions, 100
fundamental operator theorem in recursion theory, 144

$G_f$, 97
Godel numbering, 9
graph coloring problem, 91, 94, 95, 108, 113, 115, 121, 148
graph isomorphism problem, 147
graph of a function, 97
g(x,y)↓, 97
g(x,y)↑, 97

guaranteed approximation approach, 88

halting problem, 4, 9
Heine-Borel theorem, 3, 38, 42
Hierarchy of nonrecursive real numbers, 20

I(f), 75
immune sets, 18
infinitely often speedability, 126
integrals,
of partial recursive real functions, 43
of polynomial time computable functions, 74
of recursive real functions, 42
intermediate value theorem of recursive real functions, 43-44
intractable problems, 88

job scheduling problems, 88
join of two sets, 9
jump of a real number, 20
jump of a set, 20
k, 4
k-graph coloring problem, 121-122
k-language recognition problem, 122
knapsack problem, 88, 94, 101, 113, 148
k=\mathcal{R}, 122

language, 92
language recognition problems, 95
left hierarchy, 21
left recursively enumerable in, 20
left recursively enumerable real numbers, 16
length of a dyadic number, 4
length of a string, 97
lg(x), 97
linear programming problem, 147
locally polynomially bounded modulus, 59
L_x, 5

m, 82
m, 86
\leq_m, 4
\cong_m, 4
\equiv_R, 16
\leq_m^R, 16
many-one equivalence, 4
many-one reducibility, 4, 104
MAX(f), 78
maximum clique problem, 91, 94, 108, 115, 149
maximum cut problem, 149
maximum satisfiability problem, 108, 115, 149
maximum values,

of polynomial time computable real functions, 78
of recursive real functions, 43
m-complete in left r.e. numbers, 16

\bar{M}_i, 127
\hat{\bar{\mathcal{M}}}_i, 127
Minimum clique cover problem, 108, 149
modulus of continuity, 41, 51, 105
M^\varphi(n), 30
M^\varphi(n)^\uparrow, 30
M^\varphi(0^n), 31
multiplication, 27
\( M^{x}(n) \), 30

N, 3

NAP, 102

no state of an oracle Turing machine, 29

nondeterministic Turing machines, 92

nowhere differentiable functions, 70

polynomial time computable, 70

NP, 78, 93

NP*, 96

NP-complete, 80, 85, 93, 147

NP-hard, 80, 85, 88, 93, 94

NP-hard optimization problems, 88, 94

NP*(NP), 96

"NP = PP?", 120

NTIME(\( \mathcal{C} \)), 93

NTIME(\( \mathcal{T} \)), 92

\( \leq_{1} \), 4

\( \equiv_{1} \), 4

\( \leq_{R} \), 16

1-complete in left r.e. numbers, 16

one-one equivalence, 4

one-one reducibility, 4

ONTM, 96

open sets, 3

optimization problems, 106

optimal measure of, 106

optimal solutions of, 107

oracle nondeterministic Turing machines, 96

oracle Turing machines, 29, 92

function-, 29

multi-, 30

set-, 29

two-, 30

OTM, 29

P, 90, 93

P*, 96

\( \leq^{P} \), 110

\( \leq_{P}^{a} \), 105
polynomially bounded modulus, 51
\leq_1^p, 113
Post’s simple set, 132
power series, 61, 62, 77
radius of convergence of, 62
PP, 118
"P=PP?", 118
p=productive functions, 128
p=productive sets, 127–128
"P=PSPACE?", 75
PR, 46
pA, 86
PR[1], 65
primality problem, 90, 91, 119, 147, 149
primitive recursive functions, 3
primitive recursive real numbers, 6
PR^NP, 78
probabilistic algorithms, 90, 116
probabilistic approach, 89
probabilistic Turing machines, 91, 116
computation of, 116–117
functions computed by, 117
time complexity of, 116
probability distribution, 89
projection of a function, 97
proper-\Delta^p_2, 96
proper-\Delta^p_2*, 99
PRSPACE, 76
\leq_s^p, 113
\Delta^p_s, 136
pseudo-creative sets, 10
\phi_n, 9, 47
p-simple sets, 92
PSPACE-complete, 77, 85, 147
p-sparse sets, 92, 131
\leq^p_T, 93
PTM, 116
\leq^p_{tt}, 93
\leq^p_{tt}-complete sets, 92, 131, 132
quadrature formulas, 78
query state of an oracle Turing machine, 29
query tape, 29
ranges of partial recursive real functions, 29, 33, 43
real closed field, 26
reasonable probabilistic algorithms, 121
recursive convergence, 5
recursive enumeration, 8
recursive functions, 3, 95
recursive in, 7
recursive limit of recursive reals, 44
recursive measure theory, 85, 87
recursive real functions, 29, 30, 31
different formulations of, 31
two-dimensional, 32
recursive real numbers, 5, 6
complexity of, 45
recursive real sequences, 7
recursive relations, 26
recursive total variation, 44
recursively enumerable Dedekind cuts, 7
recursively enumerable real numbers, 15
left, 16
right, 16
recursively piecewise linear functions, 52
recursively uniform convergence, 52-53
reducibility, 3, 86
strong, 95
relative recursiveness, 20
relatively continuous functions, 104
right hierarchy, 22
right recursively enumerable in, 20
right recursively enumerable real numbers, 15
roots, 54
of analytic functions, 65, 69
of polynomial time computable functions, 55
of polynomials, 66-69
of recursive real functions, 55
\( R_x \), 5
\( \leq_s \), 136
S-complete, 130
set covering problem, 108, 149
set packing problem, 108, 150
$\Sigma$, 92
$\Sigma^A_n$, 21
$\Sigma^A_n$-form relations, 21
$\Sigma^e_n, 21$
$\Sigma^e, 22$
$\Sigma^P_n^*$, 96
$\Sigma^r_n, 22$
simple sets, 3
$S_M$, 45
$\leq_{sp}$, 112
space complexity, 45
strong hierarchy theorem, 22
structure preserving reducibility, 91, 110
Sturm sequences, 67-70
subcreative sets, 127
sum of subsets problem, 108, 111, 113, 115, 150

$\tau$, 86
$\leq_T$, 4

$\tau$, 4

T-complete in left r.e. numbers, 16
time complexity, 45

of computable real functions, 48-50
of discontinuous functions, 81
of nondeterministic Turing machines, 92
of oracle Turing machines, 48-49
of partial recursive real functions, 49-50
of probabilistic Turing machines, 116-117
of recursive real functions, 46

TIME$_C[0,1]^\langle\in\rangle$, 50
TIME$_C[0,1]^\langle T\rangle$, 50
TIME$_R^\langle\in\rangle$, 46
TIME$_R^\langle T\rangle$, 46
$T_M$, 45
$\leq_T^R$, 16
$\leq_T^R$ incomparable pair of left r.e. real numbers, 17
traveling salesman problem, 91, 94, 102, 113, 150
truth table equivalence, 4
truth table reducibility, 4, 16
\( \leq_{tt} \), 4
\( \cong_{tt} \), 4
\( \leq_{tt}^{R} \), 16
Turing equivalence, 4
Turing machines, 3
with oracles, 29
Turing reducibility, 4, 103
2-partition problem, 108, 150

uniform distribution, 137-138
uniform method of computing roots of a polynomial, 66
uniformly continuous functions, 41

VPP, 119
"VPP=NP?", 119-120
weak hierarchy theorem, 22
worst case analysis, 94
worst case performance of approximation algorithms, 94

|\( x | \), 97
|\( x' \), 20
|\( x^{(0)} \), 21
|\( x^{(n)} \), 21

yes state of an oracle Turing machine, 29
PART A

Computational Complexity of Real Functions

Introduction

Since Turing introduced the concept of computable (real) numbers in 1937 [25], many different directions have been taken in the study of constructive analysis (see, for example, [1, 5, 11, 22, 24]). Recursive function theory is often used as a setting for the study of effective computability of real numbers and real functions [7, 13, 17, 24], resulting in major and basic contributions in this area: namely, the formulation of computable real numbers, the comparison of different notions of computability of real numbers, the study of relationship between computability and continuity, and the recursive measure theory, to name a few. Methods of recursive function theory were used to discuss the problems mainly at the recursive and primitive recursive levels. However, as recent research in the area of computational complexity has drifted from effective computability to efficient (or, polynomial level) computability, the consideration of polynomial level complexity of real computations has become more interesting and important. In this part of the thesis, we investigate this topic using recursive function theory and other techniques and notions which have been successfully used in the study of the polynomial level complexity of combinatorial
computations.

The subjects of this part include the careful comparison of different formulations of computable real numbers, the extension of the range of computable real functions to partial functions, and the relation between time complexity and analytical properties of real functions. The approach we will take is somewhat informal. All results from classical analysis and recursive function theory are usable. Most of the proofs and the algorithms are written informally in English. This is not a part of intuitionistic mathematics, nor a logically independent theory of constructive analysis. The goal here is to gain some insight into the computation of real functions in "real" computers.

In chapter 1 we first give several ways of defining computable real numbers and compare them at the recursive, primitive recursive, and recursively enumerable levels. Then we borrow the notion of reducibility from recursive function theory to build a hierarchy of real numbers. Some well-known results are no longer true. In particular, there are no "creative" real numbers, nor "simple" real numbers. We do not even know whether there exists a many-one complete real number in each class of the hierarchy. Then, computable functions are introduced. Computable functions may not be total, but must be continuous on their domains. The domains and ranges of computable real functions are characterized by their topological properties.

In chapter 2, complexity of computable functions is defined by utilizing function-oracle Turing machines. Multiple oracle Turing
machines with mixed function and set oracles are also used. Basic properties of the functions in some interesting classes are shown. Then, we will ask and try to answer some basic questions: Are the roots of an easily computable function easily computable? Can an easily computable function be nondifferentiable? If it is differentiable, is its derivative easily computable? How about the integral? the maximum value? For the last two questions, only partial answers are found. In brief, the questions are closely related to the recent most famous $P=NP?$ problem.

Finally, in conclusion, we point out some interesting open questions and suggest some different approaches. In particular, we ask "What is the time complexity of a step function?" and a reasonable approach is proposed. This could be another new and interesting area and needs further investigation.

In the following we present basic definitions and notation which are used in part A. We expect that the readers are familiar with the basic definitions and terminology in the theory of recursive functions, including recursive functions, primitive recursive functions, Turing machines, creative sets, simple sets, and reducibilities. We also expect that the readers have some knowledge of real analysis such as open sets, continuity, analyticity, and the Heine-Borel theorem. Some knowledge of the "$P=NP?$" problem will be necessary for the later part of the chapter 2. The terminology and notation in this part will follow that of Rogers' [19] and the convention of real analysis. N is the set of all nonnegative integers.
$Z$ is the set of all integers. $R$ is the set of all real numbers.
$C$ is the set of all complex numbers. We define $D$ as the set of all
"dyadic" numbers, i.e.,
$$D = \{ m/2^n \mid m \in Z, n \in N \}.$$  
For each $n \in N$, $D_n$ is the set of all dyadic numbers with $\leq n$
binary digits in the fraction part, or, $D_n = \{ m/2^n \mid m \in Z \}$.  
We say a number $d$ in $D$ is of length less than or equal to $n$ (in
binary form) if it is in $D_n$. There is an efficient one-to-one
 correspondence between $N$ and $D$, we call it $\tau : N \rightarrow D$. Sometimes
we will say that a set $D' \subseteq D$ is recursive and actually mean that
$\tau^{-1}(D') \subseteq N$ is a recursive set. $K$ will denote the halting problem.
We will use $A, B, C$ to denote sets of integers; $S, T, X, Y$ to denote
sets of real numbers; $\phi, \psi$ to denote number-theoretical functions;
$f, g$ to denote real functions; $m, n, i, j, k$ to denote integers;
x, y, z to denote real numbers; and $d, e$ to denote numbers in $D$.
$\leq_m, \leq_1, \leq_{tt}$, and $\leq_T$ will denote many-one, one-one, truth table,
and Turing reducibilities, respectively, and $\equiv_T$ will denote r-
equivalence for $r = m, l, tt, o r T$. Other definitions such as oracle
Turing machines, and time complexity will be explained later in the
context when they appear the first time.
Chapter I
Computable Real Functions

1.1. Computable Real Numbers

Intuitively, a real number \( x \) is computable if

(i) there is a sequence of rational numbers converging to \( x \) and the sequence is computable; or,
(ii) the Dedekind cut of \( x \) is computable; or,
(iii) the binary representation of \( x \) is computable.

We formulate those three notions in the following. We first give some notation.

**Notation.** Let \( x \) be a real number, \( 0 \leq x \leq 1 \).

(i) We say that the function \( \varphi \) recursively converges to \( x \) and write

\[ A(x, \varphi) \text{ if } \varphi(n) \in D_n \text{ and } |\varphi(n) - x| \leq 2^{-n} \text{ for all } n \in \mathbb{N} \] [6].

(ii) Let \( L_x = \{ d \in D \mid d < x \} \), and

\[ R_x = \{ d \in D \mid d > x \} \).

(iii) Let \( B_x = \{ n \in \mathbb{N} \mid \text{the nth digit of the binary representation of } x \text{ is } 1 \} \).

If we use the word "recursive" for "computable", then these three notions are equivalent.

**Definition 1.1A.** A real number \( x, 0 \leq x \leq 1 \), is recursive if there is a recursive function \( \varphi \) such that \( A(x, \varphi) \).
**Definition 1.1B.** A real number $x$, $0 \leq x \leq 1$, is recursive if $L_x$ is recursive. (Actually, we mean that $\tau^{-1}(L_x)$ is recursive).

**Definition 1.1C.** A real number $x$, $0 \leq x \leq 1$, is recursive if $B_x$ is recursive.

**Theorem 1.1.** [15, 17, 18] Definitions 1.1A, 1.1B, and 1.1C are equivalent.

There are only countably many recursive real numbers because there are only countably many recursive functions. However, they include all rational numbers, all algebraic numbers, and many other transcendental numbers including $e$ and $\pi$.

Although definitions 1.1A, 1.1B, and 1.1C are equivalent, there do exist some basic differences. There exist primitive recursive operators which uniformly compute the Cauchy sequence representation of a real number $x$ from its binary representation $B_x$ and $B_x$ from $L_x$. However, there is no recursive operator which can uniformly compute $B_x$ or $L_x$ from the Cauchy sequence representation of $x$ [7, 13]. Also, if we replace in the definitions 1.1A, 1.1B, and 1.1C the recursive function $\varphi$ and recursive sets by primitive recursive ones, we obtain three nonequivalent definitions of primitive recursive real numbers [24]. From the above, it seems that the definition A (that $x$ is in class $\mathcal{C}$ (recursive, primitive recursive, etc.) if there exists a function $\varphi$ in $\mathcal{C}$ such that $A(x, \varphi)$) is the most comprehensive one. In particular, the set of primitive recursive real
numbers under definition A constitutes an arithmetically closed field but the sets defined by the other two notions are not even rings [24]. Mostowski [13] considered computable sequences of real numbers and obtained the result that the three definitions of recursive real sequences are not equivalent. Myhill [15] made some observations on the recursively enumerable level Dedekind cuts. He called them definable half-sections and compared them with recursive real numbers. Another characterization of recursive real numbers is that \( x \) is recursive if and only if there exist primitive recursive functions \( \varphi_1, \varphi_2 \) and \( \varphi_3 \) such that for all \( n \in \mathbb{N} \),
\[
| x - \varphi_1(n)/\varphi_2(n) | < 1/\varphi_3(n), \text{ and } \varphi_3(n) \not\to \infty [12].
\]
In the following, we compare the three notions at the recursively enumerable (r.e.) level.

First we state a general form of theorem 1.1.

**Definition 1.2.** Let \( x \) be a real number, \( 0 \leq x \leq 1 \), and \( S \subseteq \mathbb{N} \). We say that \( x \) is recursive in \( S \) if there is a function \( \varphi \), recursive in \( S \), and \( A(x, \varphi) \).

**Theorem 1.2.** The following are equivalent.

(i) \( x \) is recursive in \( S \).

(ii) \( L_x \) is recursive in \( S \).

(iii) \( B_x \) is recursive in \( S \).

**Proof.** The proof here is essentially the same proof as that of theorem 1.1 (see [18]). We informally sketch the proof of (i) \( \Rightarrow \) (ii).
Assume that \( \varphi \) is recursive in \( S \), and \( A(x, \varphi) \).

**Case 1.** \( x \in D \).

Then \( L_x = \{ d \in D \mid d < x \} \) is a recursive set, and hence is recursive in \( S \).

**Case 2.** \( x \notin D \).

For each \( d \in D \), we generate \( \varphi(n) \), for \( n = 1, 2, \ldots \), until we get \( m \) such that \( |d - \varphi(m)| \geq 2^{-m} \). It is clear that \( d < \varphi(m) \iff d < x \).

Also, such an \( m \) exists since \( d \neq x \) and \( \varphi(n) \) recursively converges to \( x \). So, this is a procedure for \( L_x \) which is recursive in \( S \). \( \square \)

Now, we replace recursiveness by recursive enumeration.

**Theorem 1.3.** For all real numbers \( x \), \( 0 \leq x \leq 1 \), if \( B_x \) is r.e., then \( L_x \) is r.e. The converse is false.

**Proof.** Assume that \( x \notin D \) and \( B_x \) is r.e. We use the following algorithm to accept all \( d < x \) and only those numbers.

**Stage 0.** Set \( \text{sum} \) equal to 0.

**Stage n+1.** Generate a new element \( k \) from \( B_x \). Add \( 2^{-k} \) to \( \text{sum} \).

If \( \text{sum} > d \), then accept \( d \); otherwise, go to the next stage.

End of algorithm.

Now, if \( d < x \), then \( (\exists m) \sum_{n=1}^{m} \chi_{B_x} (n)/2^n > d \). At some stage when all elements in \( B_x \cap \{1, 2, \ldots, m\} \) have been generated, \( \text{sum} \) will become greater than \( \sum_{n=1}^{m} \chi_{B_x} (n)/2^n \). At this stage, \( d \) will be accepted.
If \( d > x \), then \((\forall m)\ d > \xi_{x}^{m} \sum_{n=1}^{m} \chi_{B}(n)/2^{n}\), and hence the algorithm will never accept \( d \).

To show the converse false, we construct a set \( B \) which is not r.e. but the set \( L = \{ d \in D \mid d < \sum_{n=1}^{\infty} \chi_{B}(n)/2^{n}\} \) is r.e.

Let \( K \) be the set of the "halting problem". I.e. \( K = \{ n \mid \phi_{n}(n) \text{ converges} \} \) where \( \phi_{n} \) is the \( n \)-th partial recursive function in some fixed Gödel numbering of the partial recursive functions. Let

\[
B = K \cup \{ 2n \mid n \in K \} \cup \{ 2n + 1 \mid n \notin K \}.
\]

\( B \) is not r.e. since \( K \leq_{m} B \). Let \( y = \sum_{n=1}^{\infty} \chi_{B}(n)/2^{n} \).

To see that \( L_{y} \) is r.e., we construct the following algorithm which uses some fixed enumeration of \( K \).

Let \( d = m/2^{n} \) be given. Assume that \( n \) is even, and \( n = 2n' \) (otherwise, consider \( d' = 2m/2^{n'+1} \)).

**Stage \( k \).** Perform \( k \) steps in the enumeration of \( K \). Let \( J_{k} \) be the set of numbers generated in \( k \) steps. Form the sum

\[
c_{k} = \sum_{j \in J_{k}, j \leq n'} 2^{-(2j-1)} + \sum_{j \notin J_{k}, j \leq n'} 2^{-2j}.
\]

If \( d < c_{k} \), then accept \( d \); otherwise go to the next stage.

**End of algorithm.**

Note that \( c_{k} < y \) for all \( k \). So, all accepted \( d \) are in \( L_{y} \).

On the other hand, for each \( n \), there is a \( k \) such that

\[
c_{k} = \sum_{j=1}^{n} \chi_{B}(j)/2^{j}.
\]

so, if \( d \in D_{n} \) then \( d \) will be accepted by stage \( k \).
Theorem 1.3 tells us that the two notions $B_x$ and $L_x$ are not the same at the level of r.e. sets. It seems worthwhile to study their relationship a little further. First, note that all r.e. sets may be used as $B_x$ to define $x$, but this is obviously not true for $L_x$. Actually, $L_x$ must be "pseudo-creative".

**Definition 1.3.** [19] A set $A \subseteq \mathbb{N}$ is pseudo-creative if

(i) $A$ is r.e.,

(ii) $(\forall B, \text{r.e.}) \left[ B \subseteq A \Rightarrow (\exists C, \text{r.e.}) \left[ C \text{ is infinite } \& C \subseteq A \setminus B \right] \right]$, and

(iii) $A$ is not creative.

**Theorem 1.4.** Let $x$ be a real number, $0 \leq x \leq 1$. If $L_x$ is r.e. but nonrecursive, then $L_x$ is pseudo-creative.

**Proof.** We first show the condition (ii) in definition 1.3.

Assume that $B \subseteq D$ is r.e. and $B \subseteq R_x = L_x$. We claim that

$$(\exists d_1)(\forall e \in B) \left[ x < d_1 < e \right].$$

Then, we choose $d_2 \in D$ such that $x < d_2 < d_1$. So, we have found an infinite recursive set

$$C = \{d \in D \mid d_2 < d < d_1\}$$

such that $C \subseteq R_x \setminus B$.

All we need is to prove the claim.

(proof of claim) By way of contradiction, assume that

$$(\forall d \in D) \left[ d > x \Rightarrow (\exists e \in B) \left[ e < d \right] \right],$$

then $R_x = \{d \in D \mid (\exists e \in D)[e \in B \& e < d] \}$, and so, $R_x$ is r.e.

(note that $B$ is r.e. and "\(<" is recursive in $D$). But, then $L_x$ is r.e. and $L_x$ is r.e., thus $L_x$ is recursive, a contradiction.
Now, we show that $L_x$ is not creative.

By the way of contradiction, assume that $L_x$ is creative. Then $x \notin D$, and $L_x$ is completely creative (see, e.g. [19]). It means that there is a recursive function $\psi$ which maps each r.e. set $\hat{W} \subseteq D$ to a number in $\hat{W} - R_x$ or $R_x - \hat{W}$. If we choose $\hat{W}_d$ to be $D \cap [d, \infty)$, then, by the s-m-n theorem, there is a recursive function $f$ which maps each $d \in D$ to a number $f(d)$ in $\hat{W}_d - R_x$ or $R_x - \hat{W}_d$ and satisfies the condition $f(d) > d$ if and only if $d \in L_x$. Thus $L_x$ is recursive and it is a contradiction.

Formally, $\tau^{-1}(R_x)$ is completely productive. Or, there exists a recursive function $\psi$ such that for all $n$,

\[
\psi(n) \in \hat{W}_n - \tau^{-1}(R_x)
\]

or

\[
\psi(n) \in \tau^{-1}(R_x) - \hat{W}_n, \text{ where } \hat{W}_n = \text{domain}(\varphi_n).
\]

(1)

By the s-m-n theorem, there is a recursive function $s$ such that

\[
\varphi_{s(m)}(n) = \begin{cases} 0, & \text{if } \tau(n) \geq \tau(m) \\ \text{diverges}, & \text{otherwise}, \end{cases}
\]

or,

\[
n \in \hat{W}_{s(m)} \iff \tau(n) \geq \tau(m).\]

(2)

Let $f : D \rightarrow D$ be defined by $f(d) = \tau \circ s \circ \tau^{-1}(d)$.

We claim that $d < x \iff d \leq f(d)$. If this is true then $d < x$ is a recursive relation and $L_x$ is recursive. A contradiction.

(Proof of claim). Let $m = \tau^{-1}(d)$, then (2) becomes

\[
n \in \hat{W}_{s(m)} \tau^{-1}(d) \iff \tau(n) \geq d
\]

(3)

Thus we have $d < x$

\[
\Rightarrow (\forall e \in R_x)[e > d] \\
\Rightarrow (\forall e \in R_x)[\tau^{-1}(e) \in \hat{W}_{s(m)} \tau^{-1}(d)], \text{ by (3)}
\]
\[ \Rightarrow \tau^{-1}(R_x) \subseteq \hat{\psi}^{-1}(d) \]
\[ \Rightarrow \psi(s \circ \tau^{-1}(d)) \in \hat{\psi}^{-1}(d) \quad \text{by (1)} \]
\[ \Rightarrow f(d) = \tau \circ \psi(s \circ \tau^{-1}(d)) \geq d. \quad \text{by (3)} \]

Also,
\[ d \geq x \]
\[ \Rightarrow (\forall e \in D) \quad [\tau^{-1}(e) \in \hat{\psi}^{-1}(d) \Rightarrow e \geq d \geq x] \quad \text{by (3)} \]
\[ \Rightarrow \hat{\psi}^{-1}(d) \subseteq \tau^{-1}(R_x) \]
\[ \Rightarrow \psi(s \circ \tau^{-1}(d)) \notin \hat{\psi}^{-1}(d) \quad \text{by (1)} \]
\[ \Rightarrow f(d) = \tau \circ \psi(s \circ \tau^{-1}(d)) < d. \quad \text{by (3)} \]

Now, we use reducibilities to compare \( L_x \) and \( B_x \), and show that \( B_x \) and \( L_x \) are \texttt{tt}-equivalent but independent under many-one reducibility.

**Theorem 1.5.** For any real number \( x \), \( 0 \leq x \leq 1 \), we have

(i) \( B_x \equiv_{\text{tt}} L_x \);

(ii) for any function \( \varphi \), if \( A(x, \varphi) \), then \( B_x \preceq_T \varphi \); and

(iii) there exists a \( \varphi \) such that \( A(x, \varphi) \) and \( \varphi \preceq_T B_x \).

**Proof.** The proofs of (ii) and (iii) are almost the same as those in theorems 1.1 and 1.2. Actually, theorems 1.1 and 1.2 are corollaries of this theorem.

(i): When \( B_x \) is available as an oracle, we need only to ask \( n \) questions: "1 \( \in \) \( B_x \)?", "2 \( \in \) \( B_x \)?", ..., "\( n \in B_x \)?" in order to decide whether \( d = m/2^n \leq x \) or not. So, \( L_x \preceq_{\text{tt}} B_x \).

In order to decide whether \( n \in \), we also need only to ask once: "1/2^n \( \in \) \( L_x \)?, 2/2^n \( \in \) \( L_x \)?, ..., 2^n/2^n \( \in \) \( L_x \)?". Then, choose the largest
m such that $m/2^n \in L_x$ and accept $n$ if and only if $m$ is odd. So, $B_x \leq_{tt} L_x$.

Theorem 1.6. (i) There exists a real number $x$, $0 \leq x \leq 1$, such that $B_x$ is r.e. and $B_x \not\leq_m L_x$.

(ii) There exists a real number $y$, $0 < y < 1$, such that $L_y \not\leq_m B_y$.

(iii) There exists a real number $z$, $0 < z < 1$, such that $B_z$ is r.e. and $L_z \not\leq_1 B_z$.

Proof. (i) and (iii). Since we can have any r.e. sets as $B_x$'s but only pseudo-creative sets as $L_x$'s, we just choose $B_x$ to be creative and $B_z$ to be simple. Then $B_x \not\leq_m L_x$ and $L_z \not\leq_1 B_z$.

(ii). We will construct $B = B_y$ in stages, and during the construction, at each stage $n$, we choose an $m$ such that

$$[\varphi_n(m) \in B_y \& \tau(m) \not\in L_y] \lor [\varphi_n(m) \not\in B_y \& \tau(m) \in L_y].$$

so that $\varphi_n$ does not reduce $L_y$ to $B_y$.

Stage 0. Let $0 \in B_0$, $k_1 = 0$, and assume that $\varphi_0 = \varnothing$.

Assume that before stage $n$, we have decided the memberships of $0, 1, \ldots, k_n$ in $B$ or $\overline{B}$. Call them $B_n$ and $B_n'$, i.e.,

$$B_n = B \cap \{0, 1, \ldots, k_n\}, \quad \text{and} \quad B_n' = \overline{B} \cap \{0, 1, \ldots, k_n\}.$$ Let

$$y_n = \sum_{j=1}^{k_n} \chi_{B_n}(j)/2^j.$$  

Stage $n$. Let $d_n = y_n + 2^{-(k_n+2)}$, and $m = \tau^{-1}(d_n)$.

Case 1. $\varphi_n(m)$ diverges.

Go to the next stage.

Case 2. $\varphi_n(m)$ converges.
Subcase 2.1. \( \varphi_n(m) \leq k_n \) and \( \varphi_n(m) \in B_n \).

Then let \( k_{n+1} = k_n + 1 \), \( B_{n+1} = B_n \),
\( B_{n+1}^* = B_n^* \cup \{k_n+1, k_n+2, k_n+3\} \). Go to the next stage.

Subcase 2.2. \( \varphi_n(m) \leq k_n \) and \( \varphi_n(m) \in B_n^* \).

Then let \( k_{n+1} = k_n + 1 \), \( B_{n+1} = B_n \),
\( B_{n+1} = B_n \cup \{k_n+1\} \). Go to stage \( n+1 \).

Subcase 2.3. \( \varphi_n(m) = k_n + 1 \).

Then let \( k_{n+1} = k_n + 1 \), \( B_{n+1} = B_n \),
and \( B_{n+1}^* = B_n^* \cup \{k_n+1, k_n+2, k_n+3\} \).
Go to the next stage.

Subcase 2.4. \( \varphi_n(m) \geq k_n + 2 \).

Then let \( k_{n+1} = \varphi_n(m) \),
\( B_{n+1} = B_n \cup \{k_n+1\} \), and
\( B_{n+1}^* = B_n^* \cup \{k_n+2, k_n+3, \ldots, \varphi_n(m)\} \).
Go to the next stage.

End of algorithm.

Note first that we require the information whether \( \varphi_n(m) \) converges or not and hence \( B_y \) is not r.e. Also note that for all \( n \),
\( y_n < y < y_n + 2^{-k_n} \).

Now, for all \( n \), \( \varphi_n \) is total, we claim that \( \varphi_n \) does not reduce \( L_y \) to \( B_y \). For each \( n \), \( d_n \) will be the evidence.

In subcase 2.1, \( \varphi_n(\tau^{-1}(d_n)) \in B_n \), but, \( y_{n+1} = y_n \)
\( \Rightarrow d_n = y_n + 2^{-(k_n+2)} > y_{n+1} + 2^{-(k_n+3)} = y_{n+1} + 2^{-k_{n+1}} > y. \)

In subcase 2.2, \( \varphi_n(\tau^{-1}(d_n)) \notin B_n \), but
\[ d_n = y_n + 2^{-k+2} \leq n < y_n + 2^{-k+1} = n + 1 < y. \]

Subcase 2.3 is similar to subcase 2.1 and subcase 2.4 is similar to subcase 2.2.

Note also that the membership of \( \varphi_n(\mathbf{y}^{-1}(d_n)) \) is determined at stage \( n \) and will never be changed and so the later stages will not affect the above proof.

Now, for any \( n \), \( \varphi_n \) can not be the reduction function for \( L_y \) to \( B_y \). So, \( L_y \nleq_m B_y \). \( \Box \)

We don't know whether there is an r.e. set \( B_y \) such that \( L_y \nleq_m B_y \).

1.2. Recursively Enumerable Real Numbers

It is now natural to define recursively enumerable real numbers using the \( L_x \) notion and to borrow the results about \( L_x \) from recursive function theory to study the structure of noncomputable real numbers. It may seem that this kind of definition will not lead in any interesting new directions different from the results in recursive function theory. However, there are several reasons to do so: firstly, not all r.e. sets can be left cuts of r.e. real numbers (theorem 1.4) so that some different properties of real numbers are expected; secondly, we can consider the arithmetical operations on real numbers; and, finally, the result in section 1.5 characterizing the domains of computable real functions shows the significance of this approach.
Definition 1.4. A real number $x$, $0 \leq x \leq 1$, is called left recursively enumerable (left r.e.) if $L_x$ is r.e. and is called right r.e. if $R_x$ is r.e.

We can list some basic properties immediately.

Theorem 1.7. There exists a left r.e. nonrecursive real number.

Proof. Let $x$ be such that $B_x$ is r.e. but nonrecursive. By theorem 1.3, $L_x$ must be r.e. and by theorem 1.1, $L_x$ is not recursive. []

Theorem 1.8. [15] A real number $x$ is recursive if and only if $x$ is both left r.e. and right r.e.

Proof. Immediate. []

Theorem 1.9. [15] A real number $x$ is left r.e. if and only if $-x$ (or, to make it in $[0,1]$, $1-x$) is right r.e.

Proof. Immediate. []

Definition 1.5. Let $x$ and $y$ be two real numbers, $0 \leq x, y \leq 1$. We say that $x$ is one-one (many-one, truth table, Turing) reducible to $y$ and write $x \leq^R_y (x \leq^R_m y, x \leq^R_{tt} y, x \leq^R_T y)$ if $L_x$ is one-one (many-one, truth table, Turing) reducible to $L_y$.

Definition 1.6. A real number $x$, $0 \leq x \leq 1$, is l-(m-, tt-, T-) complete in left r.e. real numbers if

(i) $x$ is left r.e., and

(ii) $(\forall y, \text{left r.e.}) [ y \leq^R_1 x ( y \leq^R_m x, y \leq^R_{tt} x, y \leq^R_T x ) ]$. 
Lemma 1.1. (i) If \( L_x \) is \( l-(m-, \text{tt-}, T-) \) complete in r.e. sets, then \( x \) is \( l-(m-, \text{tt-}, T-) \) complete in left r.e. real numbers.

(ii) If \( x \) is \( \text{tt-}(T-) \) complete in left r.e. real numbers, then \( L_x \) is \( \text{tt-}(T-) \) complete in r.e. sets.

Proof. (i) is evident.

(ii) Assume that for all left r.e. \( y \), \( y \leq_{tt}^R x \). Then, for some \( y \), \( B_y = K \), and \( y \leq_{tt}^R x \). Now, \( B_y \equiv_{tt} L_y \) implies that \( K = B_y \leq_{tt} L_x \), and that \( L_x \) is \( \text{tt-complete} \) in r.e. sets.

The proof is similar for \( T \)-completeness. \( \square \)

Theorem 1.10. (i) There exists a \( \text{tt-complete} \) left r.e. real number.

(ii) There exists a \( T \)-complete left r.e. real number which is not \( \text{tt-complete} \).

(iii) There exists a \( \leq_{T}^R \) incomparable pair of left r.e. real numbers.

Proof. Note that \( B_x \equiv_{tt} L_x \) for all real numbers \( x \). Let \( B_x = K \), \( B_y \) be a \( T \)-complete hypersimple set, and \( B_s \), \( B_t \) be two \( T \)-incomparable r.e. sets. Then, (i) \( x \) is \( \text{tt-complete} \), (ii) \( y \) is \( T \)-complete, but is not \( \text{tt-complete} \), and (iii) \( s \) and \( t \) are \( \leq_{T}^R \) incomparable left r.e. numbers. \( \square \)

Since r.e. \( L_x \)'s must be recursive or pseudo-creative, it is not easy to see whether there exists an \( m \)-complete left r.e. real number or not. We have not solved this problem, nor the question of existence of \( l \)-complete number. Let \( y \) be the left r.e. real number such that \( B_y = K \). It is quite reasonable to guess that if there
exists an \( m \)-complete number then \( y \) is such a one. In the following, we will show that if this \( y \) is \( m \)-complete, then \( y \) must be \( 1 \)-complete too. This result strongly supports the conjecture that \( m \)-completeness and \( 1 \)-completeness coincide on the set of left r.e. real numbers.

**Definition 1.7.** [19] A is a cylinder if \( A \equiv B \times N \) for some \( B \).

One important characterization of a cylinder is that

\[
[ A \text{ is a cylinder} \iff (\forall B)[B \leq_m A \Rightarrow B \leq_1 A]].
\]

We will use another characterization of a cylinder to show that \( L_y \), whose corresponding \( \mathcal{B}_y = K \), is a cylinder.

Let \( D_n^* \) be the \( n \)-th finite set under the canonical indexing\(^1\).

**Lemma 1.2.** [19] A is a cylinder if and only if there exists a recursive function \( \psi \) such that, for all \( n \),

\[
[ D_n^* \neq \emptyset \quad \& \quad D_n^* \subseteq A ] \Rightarrow \psi(n) \in A - D_n^* \quad \text{and} \quad \\
[ D_n^* \neq \emptyset \quad \& \quad D_n^* \subseteq \overline{A} ] \Rightarrow \psi(n) \in \overline{A} - D_n^*.
\]

**Theorem 1.11.** For any real number \( x \), \( 0 \leq x \leq 1 \), and \( x \notin D \), if \( B_x \) is not immune (i.e., \( B_x \) has an infinite r.e. subset), then \( L_x \) is a cylinder. (Actually, \( \tau^{-1}(L_x) \) is a cylinder).

**Proof.** Assume that \( B_x \) has an infinite r.e. subset \( C \).

\(^1\)In [19], Rogers used \( D_n \) to denote the \( n \)-th finite set, but here we change it to \( D_n^* \) because we reserve \( D_n \) to denote the set of rational numbers of "length" \( n \).
We describe an algorithm for the function $\Psi$ of lemma 1.2.

For input $n$, we effectively find all elements of $D'_n$.

Case 1. $D'_n = \emptyset$, then output 0.

Case 2. $D'_n$ is a singleton, say, $D'_n = \{m\}$.

Let $d = \tau(m)$, and assume that $|d| = k$.

Generate $C$ until we find a number $j \in C$ such that $j > k$.

Then, output $\tau^{-1}(d + 2^{-j})$.

Case 3. $D'_n$ contains more than one element.

Let $m$ be the element in $D'_n$ such that

$$d = \tau(m) = \max \{ \tau(j) \mid j \in D'_n \}.$$ 

Then, as in case 2, find $j \in C$, $j > |d|$, and output $\tau^{-1}(d + 2^{-j})$.

End of algorithm.

First note that $C$ is infinite, so, in case 2 or 3, we can always find the required $j$.

For case 2, if $d < x$, then $d + 2^{-j} > x$. If $d < x$, then

$$d \leq \sum_{n=1}^{\infty} \chi_{B_x}(n) / 2^n,$$

so, $d + 2^{-j} < \sum_{n=1}^{\infty} \chi_{B_x}(n) / 2^n$ because $j > k$. I.e. $\tau(\psi(n)) < x \iff \tau(D'_n) \subseteq L_x$.

For case 3, if $\tau(D'_n) \subseteq L_x$, then $d < x$, so, $d + 2^{-j} < x$, and hence $\tau(\psi(n)) < x$. If $\tau(D'_n) \subseteq R_x$, then $d + 2^{-j} > x$.

Thus, $\tau(\psi(n)) > x$.

Therefore $\Psi$ satisfies the condition in lemma 1.2, so $L_x$ is a cylinder. ☐
Corollary 1.11.1. For any $x$, $0 \leq x \leq 1$, $x \notin D$, if $\bar{L}_x$ is not immune, then $L_x$ is a cylinder.

Proof. A symmetric proof of the theorem 1.11 can show that $R_x$ is a cylinder. But that also means that $L_x$ is a cylinder. □

Corollary 1.11.2. If $B_y = K$, then $L_y$ is a cylinder, and hence

$$(\forall x) \left[ x \leq^R_m y \Rightarrow x \leq^R_1 y \right].$$

Proof. $B_y = K$ is r.e., so, is not immune. □

1.3. Relative Recursiveness and Hierarchy of Nonrecursive Real Numbers

We mentioned the equivalence of three notions of relative recursiveness in theorem 1.2. Now we use relative recursiveness to build a hierarchy of real numbers as an analogue of the arithmetical hierarchy in recursive function theory.

Definition 1.8. A real number $x$ is left r.e. in $y$ if $L_x$ is r.e. in $L_y$ and $x$ is right r.e. in $y$ if $R_x$ is r.e. in $R_y$.

Definition 1.9. Let $A'$ denote the jump of a set $A$ [19]. We say that a real number $y$ is the jump of $x$, if $B_y = (B_x)'$, and write $x'$ to denote the jump of $x$.

Theorem 1.12. (i) $(\forall x) x'$ is left r.e. in $x$ but is not recursive in $x$.

(ii) If $x$ is left r.e. in $y$, then $x \leq^R_T y'$.

Proof. (i) First we state a lemma as the relativized version of
Theorem 1.3.

**Lemma.** \((\forall x) B_x\) is r.e. in \(A\) \(\Rightarrow\) \(L_x\) is r.e. in \(A\).

**Proof.** The proof of theorem 1.3 carries over. \(\Box\)

Now, by definition, \(B_x\) is the jump of \(B_x\). So, \(B_x\) is left r.e. in \(B_x\) but not recursive in \(B_x\). Since \(L_x \equiv_T B_x\), \(B_x\) is left r.e. in, but not recursive in \(L_x\). By the above lemma, \(x'\) is left r.e. in but not recursive in \(x\).

(ii) \(x\) is left r.e. in \(y\)

\[\Rightarrow L_x \text{ is r.e. in } L_y, \text{ by definition}\]
\[\Rightarrow L_x \text{ is r.e. in } B_y, \text{ since } L_y \equiv_T B_y\]
\[\Rightarrow L_x \text{ is recursive in } (B_y)' = B_y',\]
\[\Rightarrow L_x \text{ is recursive in } L_y', \text{ since } L_y' \equiv_T B_y'. \Box\]

**Definition 1.10.** \(x^{(0)} = x\), \(x^{(n-1)} = (x^{(n)})'\).

Let \(\Sigma_n^A\), \(\Pi_n^A\) be the class of relations expressible by \(\Sigma_n^A\), \(\Pi_n^A\) forms (with oracle \(A\)), and \(\Delta_n^A = \Sigma_n^A \cap \Pi_n^A\), as defined in [19].

**Definition 1.11.** Let \(x\) be a real number, \(0 \leq x \leq 1\), \(n \in \mathbb{N}\).

\[\Sigma_n^{\ell,x} = \{ y \in [0,1] \mid L_y \in \Sigma_n^x \},\]
\[\Pi_n^{\ell,x} = \{ y \in [0,1] \mid L_y \in \Pi_n^x \},\]
\[\Delta_n^{\ell,x} = \Sigma_n^{\ell,x} \cap \Pi_n^{\ell,x}.\]

A real number \(y\) is in the left hierarchy in \(x\) if \(y \in \Sigma_n^{\ell,x}\) for some \(n \geq 0\).
A similar right hierarchy can be defined and is symmetric to the left hierarchy. i.e., if we let $\Sigma_n^{R,x} = \{ y \in [0,1] \mid R_y \in \Sigma_n \}$, then $\Sigma_n^{R,x} = \Pi_n^{\ell,x}$. Also, the extension of hierarchies to include constructive ordinals as indices, e.g. $\Sigma^\ell_\omega$, is possible [14].

Some basic properties of the left hierarchy are as follows.

**Theorem 1.12.** For any $x, 0 \leq x \leq 1$, $n \in N$,

(i) $(\forall x)[y \in \Sigma_n^{\ell,x} \Rightarrow 1-y \in \Pi_n^{\ell,x}]$,

(ii) $\Sigma_n^{\ell,x} \cup \Pi_n^{\ell,x} \subseteq \Delta_n^{\ell,x}$.

**Theorem 1.14.** (Strong Hierarchy Theorem)

(i) $y \in \Sigma_{n+1}^{\ell,x} \iff y$ is left r.e. in $x^{(n)}$.

(ii) $y \in \Delta_{n+1}^{\ell,x} \iff y$ is recursive in $x^{(n)}$.

**Proof.** By the strong hierarchy theorem for the arithmetical hierarchy (see [19]).

**Corollary 1.14.1.** For any $x, 0 \leq x \leq 1$, and $n \in N$, $x^{(n)}$ is $\Sigma_n^{R,x}$-complete in $\Sigma_n^{\ell,x}$.

**Proof.** By theorem 1.14.(i), if $y \in \Sigma_n^{\ell,x}$, then $y$ is left r.e. in $x^{(n-1)}$. By theorem 1.12.(ii), $y \preceq_R x^{(n)}$.

Also note that by theorem 1.2 and theorem 1.14.(ii), the three definitions of $\Delta_n$ are equivalent.

**Theorem 1.15.** (Weak Hierarchy Theorem) For any $x, 0 \leq x \leq 1$, and $n \in N$,
(i) $\Sigma_n^{l,x} \neq \Pi_n^{l,x}$ [14],

(ii) $\Delta_{n+1}^{l,x} \neq \Sigma_n^{l,x} \cup \Pi_n^{l,x}$.

Proof. (i) $x^{(n)}$ is in $\Sigma_n^{l,x}$ but $x^{(n)}$ is not recursive in $x^{(n-1)}$.
So, $x^{(n)} \in \Sigma_n^{l,x} - \Delta_n^{l,x}$. That is, $x^{(n)} \in \Sigma_n^{l,x} - \Pi_n^{l,x}$.

(ii) We show the case $x = 0$, $n = 1$. Other cases can be shown easily
by relativization of this proof.

In the following, we will describe an algorithm for $B_a$ using

$\mathcal{K}_0 = \{ \langle x, y \rangle | x \in \mathcal{W}_y \}$ as an oracle. Since $\mathcal{K}_0$ is r.e., $B_a$ is

in $\Delta_2$, and $L_a \equiv_T B_a \Rightarrow L_a$ is in $\Delta_2$. Also, during the construction

of $B_a$, we make $L_a \not= \mathcal{W}_n$, $R_a \not= \mathcal{W}_n$ for all $n$.

Assume that $\mathcal{W}_0 = \emptyset$.

Stage 0. Let $a_0 = 0$, and $0 \not\in B_a$.

Assume that after stage $n-1$, we have determined the memberships

of 0, 1, ..., $3(n-1)$ in $B_a$ or $\overline{B_a}$, and let

$$a_{n-1} = \sum_{k=0}^{3(n-1)} \chi_{B_a}(k) / 2^k.$$

Stage $n$. Let $b_{n,1} = a_{n-1} + 2^{-(3(n-1)+2)}$, $b_{n,2} = a_{n-1} + 2^{-(3(n-1)+1)}$, and

$$b_{n,3} = a_{n-1} + 2^{-(3(n-1)+1)} + 2^{-(3(n-1)+2)}.$$

Query the memberships of $\tau^{-1}(b_{n,i})$, $i = 1, 2, 3$, in $\mathcal{W}_n$, and according to the answers and following table
determine the memberships of $3n-2$, $3n-1$, and $3n$ in $B_a$
or $\overline{B_a}$.
End of algorithm.

All we did in stage \( n \) was just to prevent \( L_a \) from being equal to \( \tau(W_n) \) or \( \tau(\bar{W}_n) \). The strategy is to decide the memberships of three more elements in \( B_a \) or \( B_a^c \) such that one of \( b_n,i \) must be in \( \tau(W_n) - L_a \) or \( L_a - \tau(W_n) \) and another \( b_n,i \) is in \( \tau(\bar{W}_n) - R_a \) or \( R_a - \tau(\bar{W}_n) \).

More formally, observe that for each \( n \), \( a_n \prec a \prec a_n + 2^{-3n} \), def \( a_n' \).

So, \( L_a \subseteq L_{a_n} \subseteq L_{a_n}' \), and \( R_a \supseteq R_{a_n} \supseteq R_{a_n}' \). Now, we want to show that for each \( n \),

\[
\begin{align*}
(1) \quad & \exists d \quad d \in \tau(W_n) - L_{a_n} \quad \text{(hence, } d \in \tau(W_n) - L_a) \\
& \quad \text{or } d \in L_{a_n} - \tau(W_n) \quad \text{(hence, } d \in L_a - \tau(W_n) ) \text{, and} \\
(2) \quad & \exists e \quad e \in \tau(\bar{W}_n) - \tau(W_n) \quad \text{(hence, } e \in \tau(\bar{W}_n) - \tau(W_n)) \\
& \quad \text{or } e \in \tau(W_n) - \tau(W_n) \quad \text{(hence, } e \in \tau(W_n) - \tau(W_n) ).
\end{align*}
\]

The following table gives \( d \) and \( e \), for each of eight cases in stage \( n \):
\[
\begin{array}{cccc}
\text{in } \tau(W_n) ? & \quad & d = & e = \\
\hline
b_{n,1} & b_{n,2} & b_{n,3} & \\
\text{yes} & \text{yes} & \text{yes} & b_{n,3} & b_{n,1} \\
\text{yes} & \text{yes} & \text{no} & b_{n,2} & b_{n,3} \\
\text{yes} & \text{no} & \text{yes} & b_{n,3} & b_{n,1} \\
\text{yes} & \text{no} & \text{no} & b_{n,2} & b_{n,1} \\
\text{no} & \text{yes} & \text{yes} & b_{n,1} & b_{n,2} \\
\text{no} & \text{yes} & \text{no} & b_{n,2} & b_{n,3} \\
\text{no} & \text{no} & \text{yes} & b_{n,3} & b_{n,2} \\
\text{no} & \text{no} & \text{no} & b_{n,1} & b_{n,3}
\end{array}
\]

\(d\) and \(e\) satisfy the required conditions (1) and (2). For example, if \(b_{n,1} \in \tau(W_n)\), \(b_{n,2} \notin \tau(W_n)\), \(b_{n,3} \notin \tau(W_n)\) (the 4th row in the above table), then \(d = b_{n,2} \notin \tau(W_n)\) and \(e = b_{n,1} \in \tau(W_n)\).

From the fourth row in the table for stage \(n\), we have \(3n-2 \in B_a\), \(3n-1 \notin B_a\), and \(3n \in B_a\). Therefore,

\[
d = b_{n,2} = a_{n-1} + 2^{-(3n-2)} < a_{n-1} + 2^{-(3n-2)} + 2^{-3n} = a_n , \quad \text{or}
\]

\[d \in L_{a_n} - \tau(W_n) . \]

And,

\[
e = b_{n,1} = a_{n-1} + 2^{-(3n-1)} < a_{n-1} + 2^{-(3n-2)} + 2^{-3n} = a_n , \quad \text{or}
\]

\[e \in \tau(W_n) - R_{a_n} . \]

Other cases can be similarly checked. And so, in all cases, (1) and (2) are satisfied. Thus \(L_a \notin \tau(W_n)\) and \(R_a \notin \tau(W_n)\), for all \(n\), and therefore \(a \in \Delta^{-1} = \Sigma^{-1} \cup \Pi^{-1}\). \(\square\)
We now consider arithmetic on real numbers. Rice [17] showed that the set of all recursive real numbers forms a real closed field. A similar result holds for the set of primitive recursive real numbers [24]. We show here that for all \( n \), \( \Delta_n^\ell \) is a real closed field.

We extend the set \( \Sigma_n^\ell \), \( \Pi_n^\ell \) and \( \Delta_n^\ell \) to consider all real numbers including \( x < 0 \) and \( x > 1 \). So, \( x \in \mathbb{R} \) is in \( \Sigma_n^\ell ( \Pi_n^\ell , \Delta_n^\ell ) \) if and only if there is a \( y \in [0,1] \) such that \( y \in \Sigma_n^\ell ( \Pi_n^\ell , \Delta_n^\ell ) \) and \( (\exists m \in \mathbb{Z}) \ x = y + m \).

**Theorem 1.16.** For all \( n \in \mathbb{N} \), \( \Sigma_n^\ell \), \( \Pi_n^\ell \), and \( \Delta_n^\ell \) are closed under addition.

**Proof.** Let \( x, y \in \Sigma_n^\ell \). Then, there are recursive relations \( R \) and \( S \) such that

\[
d \in L_x \iff (\exists a_1 \in D)(\forall a_2 \in D) \ldots (Q_n a_n \in D) R(d, a_1, \ldots, a_n)
\]

and

\[
d \in L_y \iff (\exists b_1 \in D)(\forall b_2 \in D) \ldots (Q_n b_n \in D) S(d, b_1, \ldots, b_n)
\]

where \( Q_n = \emptyset \) if \( n \) is odd and \( Q_n = y \) if \( n \) is even.

So, \( d \in L_{x+y} \iff (\exists d_1 \in L_x)(\exists d_2 \in L_y) \ d \leq d_1 + d_2 \)

\[
\iff (\exists d_1)(\exists d_2)(\exists a_1)(\exists b_1)(\forall a_2)(\forall b_2) \ldots
\]

\[
(Q_n a_n)(Q_n b_n)[R(d_1, a_1, \ldots, a_n) \& S(d_2, b_1, \ldots, b_n) \& d \leq d_1 + d_2]
\]

and hence \( L_{x+y} \in \Sigma_n \), or, \( x+y \in \Sigma_n^\ell \).

Since \( x \in \Sigma_n^\ell \iff -x \in \Pi_n^\ell \), \( \Pi_n^\ell \) is also closed under addition. So is \( \Delta_n^\ell \). \( \square \)
Theorem 1.17. $\Delta_n^\ell$ is closed under multiplication.

Proof. We only show the special case $n = 2$. The other cases can be proved similarly.

First assume that $x, y \in \Delta_2^\ell$ and $x, y > 0$. Then,

$$d < x \cdot y$$

$$\Leftrightarrow [d < 0] \lor [(\exists d_1 > 0)(\exists d_2 > 0)[d_1 \cdot d_2 = d \land d_1 < x \land d_2 < y]].$$

Since $x$ and $y$ are in $\Sigma_2$, $d_1 < x$ and $d_2 < y$ can be expressed in $\Sigma_2$-form (or, $\exists \forall$ form). By appropriate modification, $d < x \cdot y$ can also be written in $\Sigma_2$-form. So, $L_{x \cdot y} \in \Sigma_2$.

Similarly, $P_{x \cdot y} \in \Sigma_2$, or, $L_{x \cdot y} \in \Pi_2$. Thus, $x \cdot y \in \Delta_2^\ell$.

Now, for $x, y \in \Delta_2^\ell$ and $x \lor y$ is not positive, we still have $|x \cdot y| = |x| \cdot |y| \in \Delta_2^\ell$ because $\Delta_2^\ell$ is closed under subtraction. So, $x \cdot y \in \Delta_2^\ell$. \]

Theorem 1.18. [14] $\Delta_n^\ell[i]$ is an algebraically closed field.

Proof. To see that it is a field, we still need to verify that

$$x \in \Delta_n^\ell \Leftrightarrow 1/x \in \Delta_n^\ell$$

Note that

$$d < 1/x \Leftrightarrow (\exists e) [d \cdot e < 1 \land e > x]$$

and

$$d > 1/x \Leftrightarrow (\exists e) [d \cdot e > 1 \land e < x].$$

So, $x \in \Sigma_n^\ell \Leftrightarrow 1/x \in \Pi_n^\ell$, and $x \in \Delta_n^\ell \Leftrightarrow 1/x \in \Delta_n^\ell$.

Therefore, by theorems 1.16, 1.17 and the above, $\Delta_n^\ell$ is a field.

In order to see that $\Delta_n^\ell$ is algebraically closed, first observe that in Rice's proof [17, 20] of the theorem that $\Delta_1^\ell$ is algebraically closed, he actually showed that there exists a uniform recursive
procedure which, for given coefficients \( a_1, \ldots, a_n \), finds a root \( z \) such that \( \sum_{i=0}^{n} a_i z^i = 0 \).

Given this fact, it is clear that if coefficients \( a_1, \ldots, a_m \in \Delta^n \) are given (say, in function-oracle form), the procedure can recursively find a root \( z \) of \( P(x) = \sum_{i=0}^{m} a_i x^i \), and so, \( z \) is recursive in \( \langle a_1, \ldots, a_n \rangle \). Or, \( z \in \Delta_n \), because \( \Delta_n \) is closed under "recursive in".

Thus, we have shown that \( \Delta_n \) is algebraically closed.

Additional results about roots of a polynomial will be included in chapter 2.

1.4. Computable Real Functions

Recursive functionals and effective operators have been used to define computable real functions [6,7,9,11,24]. One approach is to define the computable real functions as effective operators only on the set of computable real numbers [1,11]. With this definition, many surprising results appear. For example, a continuous recursive real function on a closed interval may not be uniformly continuous; a continuous function on a closed interval may not have an integral, etc. Another approach considers all real continuous functions on all real numbers [7,9]. That is, a computable real function is an effective operator on all possible total functions (not necessarily recursive) which compute real numbers. All familiar properties in
classical analysis still hold, and the interest is on the recursiveness of these properties. In this paper, we take an approach similar to the second one and use oracle Turing machines as computing models so that we can apply the complexity measure of Turing machines to the study of the computational complexity of real functions. Our definition gives not only total recursive real functions on $[0,1]$ but also partial recursive real functions. Lacombe's definition [9] also allows partial recursive real functions, but there are very few discussions of the properties of partial recursive real functions such as domains and ranges. We characterize the domains using some natural notions including the notion of left (right) r.e. real numbers.

We begin with a description of oracle Turing machine (OTM). A (set-) oracle TM is a usual one-tape TM equipped with an extra query-tape, and three special states: the query state, the yes state, and the no state. During the computation, the machine may write some symbols on the query tape or enter the query state. When the machine enters the query state, the "oracle" will examine the string on the query tape, and, in one step, answers the query by restarting the machine in the yes state or the no state according to whether the string on the query tape is in the oracle set or not. After each query, we assume that the query tape is cleaned. A function-oracle TM is similarly equipped with a query tape and a query state. When the machine enters the query state, the oracle (a function $\varphi$) will erase the string $w$ on the query tape and write down $\varphi(w)$ on the query tape and restart the machine in a new state which must be specified by the TM program before entering the query state. Note
that although we assume that the oracle can do all this in one step, the TM must take $|\varphi(w)|$ steps to read the output $\varphi(w)$ from the query tape. We will sometimes use the two-oracle (or multi-oracle) TM's by assuming two (or more) query tapes and two (or more) query states and the interpretation is the same.

**Notation.** If $M$ is an OTM with $\varphi$, a function, as an oracle and $n$ as an input, then we write $M^{\varphi}(n)$ for the output and $M^{\varphi}(n)\uparrow$ if $M$ does not halt on input $n$ with oracle $\varphi$. If an oracle $\varphi$ which recursively converges to $x$ actually computes the binary expansion of $x$ (i.e. $(\forall n) \varphi(n) \leq x < \varphi(n) + 2^{-n}$), then we call $\varphi$ the standard oracle for $x$ and write $M^{x}(n)$ instead of $M^{\varphi}(n)$.

Now, we are ready for the definition of recursive real functions. We consider only functions with domains contained in $[0,1]$.

**Definition 1.12.** Let $S \subseteq [0,1]$ and $\overline{S} = [0,1] - S$. A function $f : S \rightarrow R$ is said to be partial recursive if there is a function-oracle TM $M$ such that

(i) $(\forall x \in S)(\forall n \in \mathbb{N})(\forall \varphi)[A(x,\varphi) \Rightarrow M^{\varphi}(n) \in D_n \text{ and } |M^{\varphi}(n) - f(x)| < 2^{-n}]$, and

(ii) $(\forall x \in \overline{S})(\forall n \in \mathbb{N})(\forall \varphi)[A(x,\varphi) \Rightarrow M^{\varphi}(n)\uparrow]$.

I.e., if $A(x,\varphi)$, and $x$ is in the domain, then $M^{\varphi}$ computes a function $\psi$ such that $A(f(x),\psi)$.

**Definition 1.13.** $f : [0,1] \rightarrow R$ is (total) recursive if $f$ is partial recursive on $[0,1]$.
Intuitively, the computation of a partial recursive real function is as following. For given $x$ (or, oracle $\varphi$) and integer $n$, the TM $M$ tries to find a rational number $d$ of length $n$ such that $d$ is close to $f(x)$. During the computation, information about $x$ can be obtained from $\varphi$. In order to consistently discuss the complexity, we need a convention on the number systems. We require that all numbers in $D_n$ be written in binary form with a "binary" point, but every integer $n$ used as input for both oracle (to get $\varphi(n)$) and machine (to get $n$ digits of $f(x)$) be written in unary system over the alphabet $\{0\}$. The reason is that when we write $n$ on the query tape or the input tape we expect a response of an $n$-digit number (in $D$), and the measure for such an operation should be based on $n$ instead of $\log n$, the length of $n$ when $n$ is written in binary form. When this is to be emphasized, we write $M^{\varphi}(0^n)$ instead of $M^{\varphi}(n)$.

As we mentioned earlier, there do not exist uniform methods of finding $E_X$ or $L_X$ from a function $\varphi$, $A(x, \varphi)$. If we replace, in definition 1.12, the oracle $\varphi$ by the set $E_X$ or $L_X$, we will get a larger class of functions including some discontinuous ones, which fact contradicts the basic assertion agreed upon by constructivists that only continuous functions are computable (cf. [6,7]). However, other formulations of recursive real continuous functions can be made without using the notion of decimal representation or Dedekind cut. Grzegorczyk made a thorough study of them and showed that all formulations are equivalent [6,7].
Now, we give some examples to illustrate the computation of recursive real functions.

(i) Many continuous functions are recursive, e.g., polynomials with recursive coefficients, all elementary functions, \(|x|\), etc. (cf. theorem 1.22).

(ii) \(f(x) = 1/x\) on \((0,1]\) is partial recursive. The computation goes as following: For an oracle \(\mathcal{P}\) and input \(n > 0\), we query \(\mathcal{P}(n), \mathcal{P}(n+1), \ldots\), until we get \(d = \mathcal{P}(2m+n+2)\) such that the first \(m\) digits of \(d\) are 0's and the \((m+1)\)st digit is 1. (If \(x = 0\), then \((\forall j) \mathcal{P}(j) = 0\) and this process will not halt.) Now, perform the long division \(1/d\) to get an \((m+n+3)\)-digit number \(e\) (an \((m+1)\)-digit integer followed by an \((n+2)\)-digit rational number). Then,

\[
|e - 1/x| \leq |e - 1/d| + |1/d - 1/x| < 2^{-(n+2)} + |x - d|/(|d| \cdot |x|) < 2^{-(n+2)} + 2^{-(2m+n+2)} \cdot 2^{m} \cdot 2^{m} = 2^{-(n+1)}.
\]

Take \(e'\), the closest number in \(D_n\) to \(e\). Then output \(e'\) \(|e' - 1/x| < 2^{-n}\).

Examples (iii) and (iv) will use the following lemma.

**Lemma 1.2.** [1,17] If \(x\) and \(y\) are recursive and \(x \neq y\), then there exists an effective method of finding whether \(x < y\) or \(x > y\). Also, this method will not halt on input \((x,y)\) if \(x = y\).

(iii) Let \(x_0\) be a recursive real number in \((0,1)\).
\[
g_{x_0}(x) = \begin{cases} 
0, & \text{if } 0 < x < x_0 \\
1, & \text{if } x_0 < x \leq 1 \\
\uparrow, & \text{if } x = x_0
\end{cases}
\]
is partial recursive.

(iv) If we extend the definition to two-dimensional functions, then
\[
f(x,y) = \begin{cases} 
1, & \text{if } x < y \\
0, & \text{if } x > y \\
\uparrow, & \text{if } x = y
\end{cases}
\]
is partial recursive.

1.5. Domains, Ranges, and Continuity

We now give a characterization of the domains of partial recursive real functions.

**Definition 1.14.** [10] (i) A set \( S \subseteq [0,1] \) is called recursively open if \( S = \emptyset \) or there exists a recursive function \( \varphi : \mathbb{N} \rightarrow \mathbb{D} \) such that
\[
(\forall n \in \mathbb{N}) \left[ \varphi(2n) < \varphi(2n+1) \right] \& \ \mathbb{S} = \bigcup_{n=0}^{\infty} ( \varphi(2n), \varphi(2n+1) ).
\]

(ii) A set \( T \subseteq [0,1] \) is recursively closed if its complement in \([0,1]\) is recursively open.

Note that we directly say \( \varphi : \mathbb{N} \rightarrow \mathbb{D} \) to mean \( \tau^0 \varphi : \mathbb{N} \rightarrow \mathbb{D} \).

**Theorem 1.19.** A set \( S \subseteq [0,1] \) is recursively open if and only if there is a partial recursive real function \( f \) whose domain is \( S \).

**Proof.** (Only if): Assume that \( S \) is recursively open and
\( S = \bigcup_{n=0}^{\infty} (\varphi(2n), \varphi(2n+1)) \). We describe an OTM \( M \) to compute \( f : S \rightarrow R \) such that \( x \in S \iff f(x) \downarrow \).

For given oracle function \( \psi, A(x, \psi) \) for some \( x \), and input \( n \), the machine \( M \) works in stages.

Stage \( \langle i, j \rangle \). Compute \( \varphi(2i) \) and \( \varphi(2i+1) \). Run the effective procedure in lemma 1.3 on \((x, \varphi(2i))\) and \((x, \varphi(2i+1))\) for \( j \) steps to determine whether \( x \in (\varphi(2i), \varphi(2i+1)) \) or not.

If, in \( j \) steps, we know that \( x > \varphi(2i) \) and \( x < \varphi(2i+1)' \), then output \( 1.\overline{0} \ldots \overline{0} \). Otherwise, go to the next stage.

End of algorithm.

If \( x \in S \), there exist \( i \) and \( j \) such that \( x \in (\varphi(2i), \varphi(2i+1)) \) and the procedure of lemma 1.3 halts on both \((x, \varphi(2i))\) and \((x, \varphi(2i+1))\) in \( j \) steps. So, \( M^\psi(n) = 1 \) by stage \( \langle i, j \rangle \).

Conversely, if \( x \notin S \), then, for all \( i \), \( x \notin (\varphi(2i), \varphi(2i+1)) \).

So, at any stage \( \langle i, j \rangle \), in \( j \) steps, we will not get a conclusion that \( x \in (\varphi(2i), \varphi(2i+1)) \) and hence \( M^\psi(n) \uparrow \).

(If): Let \( f \) be partial recursive on \( S \).

If \( S = \emptyset \), just let \( \varphi = \lambda n[0] \). So, assume that \( S \neq \emptyset \) and that the OTM \( M \) computes \( f \). The following algorithm will compute \( \varphi \) such that \( S = \bigcup_{n=0}^{\infty} (\varphi(2n), \varphi(2n+1)) \).

Let \( \mu_0 = 0 \).
Stage i. Assume that $i = \langle n, m, k \rangle$, $0 \leq m \leq 2^n$.

Simulate $M$ with the standard oracle $\psi$ of $m/2^n$ on input $l$ for $k$ steps.
If it halts, then output $\varphi(2\mu_i) = m/2^n - 1/2^k$, and
$\varphi(2\mu_i + l) = m/2^n + 1/2^k$. Let $\mu_{i+1} = \mu_i + l$, and go to the next stage.
If it does not halt in $k$ steps, then let $\mu_{i+1} = \mu_i$
and go to the next stage.

End of algorithm.

To see that the algorithm works, we check the following:

(1) $\varphi$ is a total recursive function.
(Proof). Since $S \neq \emptyset$ and $S$ is open, $S \cap D$ is infinite. So,
$\varphi$ is defined at all integers and for all $\mu$, $\varphi(2\mu) < \varphi(2\mu + l)$.

(2) $(\forall \mu_i) (\varphi(2\mu_i), \varphi(2\mu_i + l)) \in S$.
(Proof). Assume that at stage $i = \langle n, m, k \rangle$, $\varphi(2\mu_i)$ and
$\varphi(2\mu_i + l)$ are computed. I.e., $\varphi(2\mu_i) = m/2^n - 1/2^k$,
$\varphi(2\mu_i + l) = m/2^n + 1/2^k$ and $M^{2^n/2}(1)$ halts in $\leq k$ steps. Then,
for all $a \in (\varphi(2\mu_i), \varphi(2\mu_i + l))$, $|a - m/2^n| < 2^{-k}$. If $\psi$ is a function such that $A(a, \psi)$, let

$$
\psi'(j) = \begin{cases} 
\text{the first } j \text{ digits of } m/2^n, & \text{if } j \leq k \\
\psi(j), & \text{if } j > k
\end{cases}
$$

then $\psi'$ also satisfies $A(a, \psi')$.

Now, let us use $\psi'$ as an oracle to compute $M^{\psi'}(1)$. The computation of $M^{\psi'}(1)$ will be exactly the same as when we use $m/2^n$ as an oracle because the latter computation halts in $k$ steps.
and so can query at most the first \( k \) digits of \( m/2^n \). But these \( k \) digits are exactly the outputs of \( \psi^*(k) \). So, \( M^{\psi^*}(1) \downarrow \) and \( a \in S \).

(3) \( S \subseteq \bigcup_{\mu=0}^{\infty} (\varphi(2\mu), \varphi(2\mu+1)) \).

(Proof). Let \( a \in S \) and \( \psi \) compute the standard oracle for \( a \). There exists an \( n \) such that \( M^{\psi}(1) \) halts in exactly \( n \) steps. Let \( \psi(n) = m/2^n \).

Then, at stage \( i = < n, m, n > \), \( M^{m/2^n}(1) \downarrow \) in exactly \( n \) steps by the same reason argued in (2) above. (Note that since \( \psi \) is the standard oracle for \( x \),

\( \{\psi(1), ..., \psi(n-1), m/2^n, m/2^n, m/2^n, ...\} \) forms the standard oracle for \( m/2^n \).) So, the algorithm will output

\( \varphi(2\mu_1) = m/2^n - 1/2^n = (m-1)/2^n \) and \( \varphi(2\mu_1+1) = (m+1)/2^n \).

But \( |a - m/2^n| < 2^{-n} \), and hence \( a \in (\varphi(2\mu_1), \varphi(2\mu_1+1)) \). \( \square \)

The argument above that if \( \varphi \) and \( \psi \) agree at first \( k \) values, then \( M^{\varphi}(n) \) halts in \( \leq k \) steps if and only if \( M^{\psi}(n) \) halts in \( \leq k \) steps is an important observation. We will use this fact to show the continuity of partial recursive real functions.

The next theorem gives more insight into the domains of partial recursive real functions.

**Theorem 1.20.** If \( S \subseteq [0,1] \) is recursively open then

(i) \( S \cap D \) is recursively enumerable, and

(ii) If \( (a,b) \) is a component of \( S \) (i.e. \( (a,b) \in S \) but \( a \notin S \), \( b \notin S \)), then \( a \) is right r. e. and \( b \) is left r. e.
Proof. Assume that \( S = \bigcup_{n=0}^{\infty} (\varphi(2n), \varphi(2n+1)) \).

(i) For any \( d \in D \), enumerate \( \varphi(2n) \) and \( \varphi(2n+1) \) until \( d \in (\varphi(2n), \varphi(2n+1)) \) is found. This is a recursive procedure to accept \( S \cap D \).

(ii) We show only that \( a \) is right r. e. A similar proof works for proving that \( b \) is left r.e.

Case 1. \( a \) is not a right endpoint of another component of \( S \).

Then, there exist \( d \) and \( e \) in \( D \) such that \( d < a < e < b \) and \([d, a] \cap S = \emptyset\). It is now clear that \( R_a \) is r. e. by observing that

\[
R_a = \left\{ d' \in D \mid (\exists e') \left[ e' \in S \cap D \land d < e' < d' \right] \right\}.
\]

Since the set \( S \cap D \) is r. e. (by (i) of the theorem), \( R_a \) is r. e.

Case 2. \( a \) is also a right endpoint of a component of \( S \). i.e. there exist \( d, e \in D \) such that \( d < a < e \) and \([d, a) \cup (a, e] \subseteq S\).

Assume that \( a \in D \). We show that \( a \) is recursive by constructing the following algorithm.

For any input \( j \), we proceed in stages.

Stage \( n \). Compute \( \varphi(0), \varphi(1), \ldots, \varphi(2n), \varphi(2n+1) \). If there exists an \( i \leq 2^j \) which satisfies the following conditions (1), (2), and (3):

1. \((\exists k_1 \leq n) \left[ \varphi(2k_1) < d < \varphi(2k_1+1) \right] \),
2. \((\exists k_2 \leq n) \left[ \varphi(2k_2) < e < \varphi(2k_2+1) \right] \), and
(3) \((\forall k_{j} \leq n) \left\{ \left[ \varphi(2k_{j}) < e \land \varphi(2k_{j}+1) > d \right] \right\} \Rightarrow \left[ \left[ \frac{i}{2^{j}} < \varphi(2k_{j}+1) < \frac{i+1}{2^{j}} \right] \right.
\lor \left( \exists k_{4} \leq n \left[ \varphi(2k_{4}) < \varphi(2k_{j}+1) < \varphi(2k_{4}+1) \right] \right]
\left. \right\} \land \left[ \left[ \frac{i}{2^{j}} < \varphi(2k_{j}) < \frac{i+1}{2^{j}} \right] \right.
\lor \left( \exists k_{4} \leq n \left[ \varphi(2k_{4}) < \varphi(2k_{j}) < \varphi(2k_{4}+1) \right] \right) \}
\right\}

then halt and output \(i/2^{j}\). Otherwise, go to the next stage.

End of algorithm.

Note that \((\exists i \leq 2^{j}) (1) \land (2) \land (3)\) is a recursive predicate which is equivalent to saying that
\([d,e] \subseteq \left( \bigcup_{k=0}^{n} (\varphi(2k), \varphi(2k+1)) \right) \cup \left( \frac{i}{2^{j}}, \frac{i+1}{2^{j}} \right)\).

If the above algorithm halts at stage \(n\), then
\([d,a] \cup (a,e] \subseteq \left( \bigcup_{k=0}^{n} (\varphi(2k), \varphi(2k+1)) \right) \cup \left( \frac{i}{2^{j}}, \frac{i+1}{2^{j}} \right).\) So, \(a \in \left( \frac{i}{2^{j}}, \frac{i+1}{2^{j}} \right)\), or \(|a - \frac{i}{2^{j}}| < 2^{-j}\) and the algorithm is correct.

Conversely, if \(a \in \left( \frac{i}{2^{j}}, \frac{i+1}{2^{j}} \right)\), then
\([d,i/2^{j}] \cup [(i-1)/2^{j}, e] \subseteq S = \bigcup_{k=0}^{\infty} (\varphi(2k), \varphi(2k+1))\).

But \([d, i/2^{j}] \cup [(i-1)/2^{j}, e]\) is a compact set and by the Heine-Borel theorem, there exists a \(n\) such that
\([d, i/2^{j}] \cup [(i-1)/2^{j}, e] \subseteq \bigcup_{k=0}^{n} (\varphi(2k), \varphi(2k+1))\).

So, by stage \(n\), the algorithm halts and output \(i/2^{j}\). \(\square\)

From the proof of case 2 of the (if) part, we have

**Corollary 1.20.1. [10]** An isolated point of a recursively closed set is recursive.
Note that the conditions (i) and (ii) in the theorem 1.20 are not sufficient to show the recursive openness of an open set $S$. This can be easily seen by the following argument of cardinality.

First observe that there are only countably many recursive open sets in $[0,1]$. So, all we need to show is that there are uncountably many open sets in $[0,1]$ satisfying conditions (i) and (ii). Let us consider the sequences $\{d_n\}$ of dyadic numbers in $D \cap [0,1/2]$ which satisfy the following conditions:

(a) $d_0 = 1/2$,
(b) $d_n > d_{n+1}$, and
(c) $d_n \to 0$.

It is obvious that there are uncountably many such sequences. For each such sequence $\{d_n\}$, there is an open set $S = (d_0, \sqrt{2}, 1) \cup \bigcup_{n=0}^{\infty} (d_{n+1}, \sqrt{2}, d_n \cdot \sqrt{2})$ satisfying the conditions (i) and (ii). In addition, different sequences $\{d_n\}$ and $\{d'_n\}$ determine different open sets. Thus we have shown that there are uncountably many open sets satisfying the conditions (i) and (ii) and hence the conditions (i) and (ii) are not sufficient to show the recursive openness of an open set.

However, it is easily seen that if there is a uniform way of computing the endpoints of all open components of $S$, then $S$ is recursively open. Whether such a uniform computation of the endpoints of open components of recursively open set $S$ always exists is left open.
Theorem 1.21. If \( S = \bigcup_{i=0}^{\infty} (a_i, b_i) \) and \((a_i, b_i)\)'s are pairwise disjoint, and there is a recursive function \( \varphi : \mathbb{N} \times D \rightarrow \mathbb{N} \) such that
\[
\{ d \in D \mid \varphi(2i, d) \downarrow \} = R_{a_i}, \quad \text{and}
\{ d \in D \mid \varphi(2i+1, d) \downarrow \} = L_{b_i},
\]
then \( S \) is recursively open.

Proof. For each \( n = \langle i, j \rangle, \ j \geq 1 \), generate \( j \) numbers from \( R_{a_i} \cap L_{b_i} \), and output \( \psi(2n) = \text{minimum of these } j \text{ numbers} \), and
\[
\psi(2n+1) = \text{maximum of these } j \text{ numbers}. \quad \square
\]

So, if \( S \) is a finite union of open intervals with left r.e. right endpoints and right r.e. left endpoints then \( S \) is a domain of some partial recursive real function.

The most important result in every approach to constructive analysis is that only continuous functions are computable. We show this along with the stronger result that all continuous functions are computable in some relative sense.

Theorem 1.22. If \( f : S \rightarrow R \) is partial recursive, then \( f \) is continuous on \( S \). Conversely, if \( f \) is continuous on \([0,1]\), then there exists a set-oracle \( A \) of integers such that \( f \) is partial recursive in \( A \).

Proof. Assume that \( f \) is partial recursive, then the argument in the proof of theorem 1.19 can be used to show that if \( |x - y| < 2^{-k} \) and \( M^\varphi(n) \downarrow \) in \( k \) steps, then for some function \( \varphi, A(x, \varphi), M^\varphi(n) \)}
works exactly the same as $M^y(n)$. So,
\[
|f(x) - f(y)| \leq |f(x) - M^o(n)| + |M^o(n) - M^y(n)| + |M^y(n) - f(y)| < 2^{\omega n} + 0 + 2^{-n} = 2^{-(n-1)}.
\]

So, $f$ is continuous. In addition, the modulus of continuity can be effectively found.

Conversely, assume that $f$ is continuous on $[0,1]$. Then $f$ is uniformly continuous. Let $m$ be the modulus function, i.e.,
\[
(\forall x, y \in [0,1]) (\forall n \in N) \quad |x - y| < 2^{-m(n)} \Rightarrow |f(x) - f(y)| < 2^{-n}.
\]

Let $A_1 = \{<d, n, e> | n \in N \land e \in D_n \land e \leq f(d)\}$ and $A_2 = \{<n, k> | n \in N \land k \leq m(n)\}$, and let $A = A_1 \cup A_2 = \{2n | n \in A_1\} \cup \{2n+1 | n \in A_2\}$.

Now consider the following algorithm:

For given $\varphi, A(x, \varphi)$, and integer $n$, from $A_2$ find $m(n+1)$, and get $d = \varphi(m(n+1))$. Then binary search $e \in D_{n+1}$ such that $<d, n+1, e> \in A_2$ & $<d, n+1, e+2^{-(n+1)}> \notin A_1$.

Output $e$.

(End of algorithm).

Then, $|e - f(x)| \leq |e - f(d)| + |f(d) - f(x)| < 2^{-(n+1)} + 2^{-(n+1)} = 2^{-n}$.

\[
\textbf{Corollary 1.22.1.} \text{ If } f \text{ is recursive on } [0,1], \text{ then } f \text{ is uniformly continuous on } [0,1]. \text{ Moreover, the modulus of uniform continuity is recursive. I.e. there exists a recursive function } m : N \to N \text{ such that } (\forall x, y \in [0,1]) (\forall n \in N) \quad |x - y| < 2^{-m(n)} \Rightarrow |f(x) - f(y)| < 2^{-n}.
\]
Proof. In the following, let $n$ be a fixed integer.

If $M^d(n) \downarrow$ in $k$ steps, then $(d-2^{-k}, d+2^{-k})$ is an open covering of $d$ such that

$$x \in (d-2^{-k}, d+2^{-k}) \Rightarrow (\exists \varphi) \left[ A(x, \varphi) \wedge M^\varphi(n) = M^d(n) \right]$$

and hence

$$|x - d| < 2^{-k} \Rightarrow |f(x) - f(d)| < 2^{-(n-1)}.$$

Now, for fixed $n$, there is a set of open coverings of $[0,1]$:

$$\{ (d-2^{-(k+1)}, d+2^{-(k+1)}) \cap [0,1] \mid M^d(n) \downarrow \text{ in } k \text{ steps} \}$$

and by the Heine-Borel theorem, there is a finite subclass of coverings which covers $[0,1]$. So, in order to find $m(n)$, all we need is to compute $M^{d_1}(n+2)$, $M^{d_2}(n+2)$, ... (where $\{d_i\}$ is an effective enumeration of $D$) and after finding each $M^{d_i}(n+2)$, which converges in $k_i$ steps, compute $(d_i-2^{-(k_i+1)}, d_i+2^{-(k_i+1)})$ and check whether

$$\bigcup_{j=1}^{i} (d_j-2^{-(k_j+1)}, d_j+2^{-(k_j+1)}) \supseteq [0,1].$$

If yes, then take $m(n) = \max \{ k_j \mid j \leq i \}$. By the Heine-Borel theorem, this algorithm will halt. Also, if $|x - y| < 2^{-m(n)}$, then there exists a $d_j$, $j \leq i$, such that $|x - d_j| < 2^{-(k_j+1)}$. Then,

$$|y - d_j| < 2^{-m(n)} + 2^{-(k_j+1)} \leq 2^{-k_j}.$$

So,

$$|f(x) - f(y)| \leq |f(x) - f(d_j)| + |f(d_j) - f(y)| < 2^{-(n+1)} + 2^{-(n+1)} = 2^{-n}.$$

Corollary 1.22.2. If $f$ is recursive on $[0,1]$, then $\int_0^1 f$ is a recursive real number.
Remark. If $f$ is a partial recursive function with domain $S$, then $\int_S f$ may not be recursive. A simple example is as follows:

Let

$$f(x) = \begin{cases} 1, & \text{if } x < a, \\ \uparrow, & \text{if } x \geq a, \end{cases}$$

where $a$ is a left r.e. but nonrecursive real number. Then $f$ is partial recursive with domain $[0,a)$, but $\int_{[0,a)} f = a$ is not recursive.

We know that a continuous function maps an interval to an interval (or, a point). So, the range of a partial recursive function must be a countable union of intervals (open, closed, or semiclosed). A result of $[9,24]$ says that a recursive function must have a recursive maximum value. It follows that the endpoints of a closed interval component or that of a closed side of a semiopen interval component of the range of a partial recursive function must be recursive. Also, we can show that a right endpoint of an open interval component which is not a left endpoint of another open side of an interval component must be left r.e. (by a similar argument of the proof of theorem 1.20).

The only case left is whether a point being both a left endpoint and a right endpoint of some open interval components of a range is recursive or not. We conjecture that it is but are not able to show it.

Besides the above mentioned results about continuity and recursive maximum value, many theorems in classical analysis have been discussed in the setting of recursive analysis $[9,10,24]$. We name some of them here. Let $f$ be a recursive function. If $x$, $y$ and $a$
are recursive reals with $x < y$, and $a$ between $f(x)$ and $f(y)$, then there is a recursive $z$, $x < z < y$, such that $f(z) = a$ [9]. The absolute maximum value of $f$ is recursive, but the points $x$ where $f(x)$ obtains maximum values may not be recursive [1,10,24]. There exists a bounded r. e. set of recursive real which does not have a recursive limit [17]. There exist recursive increasing functions on $[0,1]$ whose difference has a nonrecursive total variation, but every recursive function of recursive total variation is the difference of two recursive increasing functions [10].

It is clear from the earlier examples that there exists a partial recursive function $f$ which is not extendable to a total recursive function. Also, in [10], it is shown that there is a function $f$ defined on an open domain $P$ which is nonrecursive but the restriction of $f$ on any closed subinterval $Q$ with recursive endpoints is recursive. Thus, the fundamental operator theorem in recursion theory that every partial recursive operator can be extended to a total recursive operator on total functions is no longer true.

We will not discuss these properties of recursive real functions furthermore. Instead, we will study the relationship between computational complexity and analytical properties of real functions.
Chapter II

Polynomial Time Computable Real Functions

2.1. Time Complexity

We use the concept of complexity of Turing machines to define the complexity of recursive real numbers and recursive real functions. We treat a TM as a machine recognizing a language, i.e., a set of strings from some alphabets (or, computing a function from strings to strings if it is a transducer). For each Turing machine $M$, a time complexity function $T_M$ and a space complexity function $S_M$ are assumed. They must satisfy Blum's axioms (see, for example, [4]). The most usual interpretation is that $T_M(n)$ is the maximum number of moves $M$ takes to recognize a string of length $n$ and $S_M(n)$ is the maximum number of cells $M$ scanned to recognize a string of length $n$. In this chapter, we assume that when an integer $n$ is used as an input (to a TM or an oracle) to ask for an answer of a number in $D$ of $n$ digits, $n$ is written in unary form so that $|n| = n$ and the numbers in $D$ are written in binary form so that $|d| \leq n$ if $d \in D_n$. Note that a number $1/2$ may be of any length depending upon how it is represented. More formally, if $A(x, \varphi)$, then $\varphi$ is a function defined on $0^*$ and outputs strings in $(0+1)^* (0+1)^*$ such that $\varphi(0^n)$ represents a number $d \in D$ and $|d - x| < 2^{-n}$. 45
Definition 2.1. A recursive real number \( x \) has time complexity \( \leq T \) if there is a \( TM \) \( M \) computing a function \( \varphi \) such that \( A(x, \varphi) \) and the time complexity function \( T_M \) of \( M \) is bounded above by \( T \).

Intuitively, a recursive real number \( x \) is of time complexity \( \leq T \), if there is an effective, uniform method, which, in \( T(n) \) steps, can get \( n \) significant digits of \( x \). This is the main reason to use unary representation for input \( n \); otherwise, every real number \( x \) has at least an exponential time (and space) complexity.

Let \( T \) be a function and \( \mathcal{C} \) a class of functions.

Definition 2.2.

\[ \text{TIME}_R(T) = \{ x \in R \mid x \text{ is recursive and is of time complexity } \leq T \} \]

\[ \text{TIME}_R(\mathcal{C}) = \bigcup_{T \in \mathcal{C}} \text{TIME}_R(T) \]

\[ \text{PR} = \text{TIME}_R(\text{Poly}) \]

\[ \text{EXPR} = \text{TIME}_R(\text{Exp}) \]

where \( \text{Poly} = \{ p \mid p \text{ is a polynomial} \} \), and

\[ \text{Exp} = \{ f \mid (\exists i \geq 1) (\forall n) f(n) = 2^{in} \} . \]

A real number \( x \) is said to be polynomial (exponential) time computable if \( x \in \text{PR} (x \in \text{EXPR}) \). All rational numbers are polynomial time computable. We will see later that all algebraic numbers are polynomial time computable. \( e \) and \( \pi \) are polynomial time computable.

Of course, not every recursive real number is polynomial time computable. Let us assume that we can effectively enumerate all polynomial time bounded TM's. I.e., there exist an effective
function $U$ and a polynomial $q$ such that $U(i)$ is the code of $i$-th polynomial time computable TM and it takes only $q(i)$ steps to get $U(i)$. Let $\varphi_i$ denote the $i$-th polynomial time computable function computed by $U(i)$ so that the time complexity function for $\varphi_i$ is \[ p_i(n) = n^i + i \] (cf. [3]). Now we construct $\psi$ by the following algorithm:

\[ \psi(0) = 0 ; \quad \psi(1) = 0.0 ; \quad \psi(2) = 0.01 ; \quad \text{and for } i > 2, \]

compute $\psi(2^i)$ as follows:

First compute $\varphi_{i-2}(2^i)$. If $\varphi_{i-2}(2^i)$ is not a number in $D_{2^i}$, then let $\psi(2^i) = 0.\ldots0101\ldots01$.

same as $\psi(2^{i-1})$'s $2^{i-1}$ digits

If $\varphi_{i-2}(2^i)$ is a number in $D_{2^i}$, then get the $(2^{i-1}+1)$st to the $(2^i)$th digits of $\varphi_{i-2}(2^i)$. Let us denote the $k$-th digit of $x$ by $x_k$ and define

\[
(\psi(2^i))_k = \begin{cases} 
(\psi(2^{i-1}))_k & \text{if } k \leq 2^{i-1} \\
0 & \text{if } k > 2^{i-1} \text{ and } k \text{ is odd and } (\varphi_{i-2}(2^i))_k = 1 \\
1 & \text{if } k > 2^{i-1} \text{ and } k \text{ is odd and } (\varphi_{i-2}(2^i))_k = 0 \\
0 & \text{if } k > 2^{i-1} \text{ and } k \text{ is even and } (\psi(2^i))_{k-1} = 1 \\
1 & \text{if } k > 2^{i-1} \text{ and } k \text{ is even and } (\psi(2^i))_{k-1} = 0 
\end{cases}
\]

e.g. if $\varphi_{i-2}(2^i) = 0.\ldots10011000\ldots01$, then

\[
\psi(2^i) = 0.\ldots01100110\ldots10.
\]

same as $\psi(2^{i-1})$'s $(2^{i-1}+1)$st to $(2^i)$th

And, $\psi(2^{i-1}+1), \ldots, (2^i-1)$ are defined to be the first $2^{i-1}+1$, $(2^i-1)$ digits of $\psi(2^i)$, respectively.
End of algorithm.

It is obvious that $\psi$ computes a real number, call it $x$. The time for computing $\varphi_{i-2}(2^i)$ is less than or equal to $(2^i)^{i-2} + (i-2)$. So, it takes $q(i-2)$ steps to get the code of $\varphi_{i-2}$ and $O(2^i)$ steps to compute the second half of the binary representation of $\psi(2^i)$. The total time for computing $\psi(n)$ is

$$\leq q(\log n) + O(n^{\log n}) \leq O(2^n)$$

and so $\psi$ is exponential time computable.

We claim that $x$ is not polynomial time computable. By way of contradiction, assume that $x$ is polynomial time computable, say, $A(x, \varphi_i)$, $i > 0$. Then $A(x, \varphi_i)$ and $A(x, \psi)$ imply that

$$|\varphi_i(2^{i+2}) - \psi(2^{i+2})| < 2^{-(2^{i+2})}. \tag{1}$$

But the $(2^{i+1}+1)$th digit of $\psi(2^{i+2})$ and that of $\varphi_i(2^{i+2})$ are different. Assume, without loss of generality, that the $(2^{i+1}+1)$th digit of $\psi(2^{i+2})$ is 1 and that of $\varphi_i(2^{i+2})$ is 0. Then, in order to satisfy the condition (1), $\varphi_i(2^{i+2})$ must have 0111...1 at its $(2^{i+1}+1)$st to $(2^{i+2})$nd digits, and $\psi(2^{i+2})$ must have 1000...0 at its $(2^{i+1}+1)$st to $(2^{i+2})$nd digits. However, we know that $\psi$ has alternative digits for every two digits and the above situation is impossible. Therefore, we have directly constructed an exponential time computable real number which is not polynomial time computable.

In order to define the time complexity of real functions, we need to define the time complexity of an oracle Turing machine. The
time complexity function of an OTM is defined similar to that of ordinary TM's except that we assume that the machine needs \( n \) steps to write a string of \( n \) symbols (here, \( 0^n \)) on the query tape and 1 step to query. The machine may decide not to read all the symbols of the output from the oracle; however, if it does read all the symbols of the output on the query tape, it takes \( |\varphi(0^n)| \) (here, \( = n \)) steps to read it (cf. [23]). When multi-oracle TM's are concerned and if some oracles are functions from integers to integers, we will follow the convention to treat them as functions from \( \{0, 1\}^* \) to \( \{0, 1\}^* \).

From the proof of corollary 1.2.1, we can see a close relationship between the modulus of continuity and computation time of a function. For a partial function which is not uniformly continuous, we can easily see that there is no recursive upper bound of the computation time of the function. Moreover, even for a simple function such as

\[
f(x) = \begin{cases} 
0, & \text{if } x < 1/2, \\
\uparrow, & \text{if } x \geq 1/2,
\end{cases}
\]

there is no recursive upper bound of its computation time.

(Proof). For any recursive function \( T \) and any integer \( n \), we can find an oracle \( \varphi \) for 1/2 and an oracle \( \psi \) for \( 2^{-1}2^{-\lfloor T(n)\rfloor} \) such that \( \varphi(k) = \psi(k) \), for all \( k \leq T(n) \). So, \( M \) will not be able to distinguish \( \varphi \) from \( \psi \) in \( \leq T(n) \) steps. This means that for any \( T \), there is a point \( x \) which is so close to 1/2 that, in \( T(n) \) steps, \( M^x(n) \) does not halt. (End of proof).
Thus we will only discuss the time complexity of computable total functions. In chapter 3, we will give some suggestions as how to reasonably define the time complexity of partial or discontinuous functions.

**Definition 2.3.** If \( f : [0,1] \rightarrow R \) is recursive, then we say that the time complexity of \( f \) is \( \leq T \), if there is an OTM \( M \) which computes \( f \) with time complexity \( T_M \) bounded above by \( T \).

**Definition 2.4.** Let \( C[0,1] \) be the class of real functions continuous on \([0,1]\).

\[
\text{TIME}_{C[0,1]}(T) = \left\{ f \in C[0,1] \mid f \text{ is recursive and the time complexity of } f \leq T \right\}
\]

\[
\text{TIME}_{C[0,1]}(\mathcal{C}) = \bigcup_{T \in \mathcal{C}} \text{TIME}_{C[0,1]}(T)
\]

\[
\text{PF} = \text{TIME}_{C[0,1]}(\text{Poly})
\]

\[
\text{EXPF} = \text{TIME}_{C[0,1]}(\text{Exp})
\]

A function in \( \text{PF} \) (\( \text{EXPF} \)) is called a polynomial (exponential) time computable real function. Almost all commonly used continuous functions are polynomial time computable. For example, if a function \( f \) is analytic on \([0,1]\), i.e., \( f(x) = \sum_{n=0}^{\infty} (\frac{f(n)}{(n)!}) \cdot (x - \frac{1}{2})^n \) for all \( x \in [0,1] \), then, by computing the first \( k \) terms of its Taylor series, we can easily get an approximation of \( f(x) \) and the error is bounded by \( (f(k+1)(\bar{x}) / (k+1)! \cdot 2^{-(k+1)} \) for some \( \bar{x} \) between \( x \) and \( 1/2 \). For many functions, this is a good error bound of order \( O(2^{-k}) \) and this computation is a way of computing
the first $k$ significant digits of $f(x)$ in $p(k)$ steps for some polynomial $p$ which depends upon $f^{(k)}(x)$.

2.2. Complexity and Modulus of Continuity

If a function $f$ is polynomial time computable then there exist an OTM $M$ and a polynomial $p$ such that for all $x \in [0,1]$, in order to get $n$ significant digits of $f(x)$, $M$ does not need any information except the first $p(n)$ digits of $x$. This is because $M$ does not have time to read more than $p(n)$ digits from the oracle for $x$. This fact is reflected in the next theorem.

**Definition 2.5.** Let $f$ be a continuous function on $[0,1]$. We say that $f$ has a polynomially bounded modulus if there is a polynomial $p$ such that for all $x, y \in [0,1]$ and $n \in \mathbb{N}$,

$$|x - y| < 2^{-p(n)} \Rightarrow |f(x) - f(y)| < 2^{-n}.$$  

**Theorem 2.1.** If $f$ is polynomial time computable then $f$ has a polynomially bounded modulus. Conversely, if $f \in C[0,1]$ has a polynomially bounded modulus, then there exists an oracle set $A$ such that $f$ is polynomial time computable in $A$ (or, $f$ is in $\text{PF}(A)$).

**Proof.** Let $f$ be polynomial time computable. Let $M$ compute $f$ and $p$ be a time bound for $M$. In the proof of theorem 1.22, we showed that if $|x - y| < 2^{-k}$ and $M^v(n) \downarrow$ in $k$ steps, then $|f(x) - f(y)| < 2^{-(n-1)}$. Now, we know that $M^v(n)$ must halt in $p(n)$ steps, and hence $|x - y| < 2^{-p(n)} \Rightarrow |f(x) - f(y)| < 2^{-(n-1)}$. 

The converse is shown in a manner similar to the second part of the proof of theorem 1.22. We simply let
\[ A = \{ \langle d, n, e \rangle \mid e \in D_n \land e \leq f(d) \} \]
be the oracle. The algorithm in that proof used \( O(m(n+1)) \) steps to get \( d \) and \( O(n+1) \) steps to perform the binary search for \( e \). Since \( m \), the modulus function, is bounded by a polynomial \( p \), the whole procedure used only \( O(n+1) + O(p(n+1)) \) steps and it is a polynomially bounded procedure.]

In the following, we give another characterization of polynomial time computable real functions. It is a natural approach from the point of view of analysis.

**Definition 2.6.** A function \( f : [0,1] \rightarrow \mathbb{R} \) is recursively piecewise linear if there exists a finite number of pairs \((x_0,y_0), (x_1,y_1), \ldots, (x_n,y_n)\) of numbers in \( D \) such that

(i) \( 0 = x_0 < x_1 < \ldots < x_n = 1 \),

(ii) \( \forall i = 0, 1, \ldots, n \) \( f(x_i) = y_i \), and

(iii) \( \forall x \in [0,1] \) \( \forall i = 1, 2, \ldots, n \) \( x_{i-1} \leq x \leq x_i \)

\[ \Rightarrow (f(x) - f(x_{i-1}))(x - x_{i-1}) = (f(x_i) - f(x_{i-1}))(x - x_{i-1}) \]

We show that a polynomial time computable real function is a limit of a sequence of uniformly convergent recursively piecewise linear functions with a polynomially bounded convergent speed.

**Theorem 2.2.** A function \( f : [0,1] \rightarrow \mathbb{R} \) is recursive if and only if there exists a sequence \( \{f_n\} \) of recursively piecewise linear functions of \( 2^n \) break points such that \( f_n \) recursively uniformly
converges to \( f \) (\( f_n \xrightarrow{\text{rec}} f \)).

More formally, there exist recursive functions \( \Psi : N \times D \rightarrow D \) and \( \varphi : N \rightarrow N \) such that

(i) \( (\forall d \in D)[|d| = n \Rightarrow \Psi(n,d) = f_n(d)] \), and
(ii) \( (\forall x \in [0,1])(\forall d \in D)(\forall n \in N)[[|d| = \varphi(n) \text{ and } |x-d| < 2^{-\varphi(n)}] \Rightarrow |f(x) - \Psi(\varphi(n),d)| < 2^{-n}] \).

**Proof.** (Only if): Let \( f \) be recursive and \( M \) compute \( f \). Since \( f \) is uniformly continuous, let \( m \) be the modulus function of \( f \), i.e.,

\[
|x - y| < 2^{-m(n)} \Rightarrow |f(x) - f(y)| < 2^{-n}.
\]

From corollary 1.2.2.1, \( m \) is recursive. Assume that \( m(n) \geq n \).

Let \( f_n \) be the recursively piecewise linear function with endpoints \( \{(k/2^n, M^k/2^n(n))\}_{k=0}^{2^n} \). To see that \( f_n \xrightarrow{\text{rec}} f \), we define \( \varphi(n) = m(n+1) \), and \( \Psi(n,d) = M^d(n) \). Now, if \( |d| = n \), then \( (d, \Psi(n,d)) \) is just one of the endpoints of \( f_n \). So, (i) is satisfied. Also, if \( |d| = \varphi(n) \) and \( |x-d| < 2^{-\varphi(n)} \), then

\[
|f(x) - \Psi(\varphi(n),d)| \\
\leq |f(x) - f(d)| + |f(d) - f_\varphi(n)(d)| \\
< 2^{-m(n+1)} + 2^{-\varphi(n)} \leq 2^{-n}.
\]

(If): Assume that \( \Psi \) and \( \varphi \) exist, are recursive, and satisfy the conditions (i) and (ii) for some sequence \( \{f_n\} \) of recursively piecewise linear functions. Then, it is clear that condition (ii) that

\[
|x - d| < 2^{-\varphi(n)} \Rightarrow |f(x) - \Psi(\varphi(n),d)| < 2^{-n}
\]
gives us a method of computing \( f \). \( \square \)
Corollary 2.2.1. \( f : [0, 1] \to \mathbb{R} \) is polynomial time computable if and only if \( f \) is a limit of a sequence of recursively piecewise linear functions which converges in polynomial time (i.e., the function \( \psi \) in the theorem 2.2 is polynomial time computable and \( \varphi \) is a polynomial).

**Proof.** (Only if): Assume that \( f \) is polynomial time computable.

From theorem 2.1, \( m \) is a polynomial and hence \( \varphi \) is a polynomial. Now, for given \( n \) and \( d \), \( M^d(n) \) takes only \( p_f(n) \) steps (where \( p_f \) is the time complexity function of \( f \)) and so, \( \psi \) is also polynomial time computable.

(If): Assume that \( \psi \) is polynomial time computable and \( \varphi \) is a polynomial. Let \( p_\psi \) and \( p_\varphi \) be the polynomial time bounds for computing \( \psi \) and \( \varphi \), respectively. Then, for given \( \psi' \), \( A(y, \psi') \), and \( n \), in order to find \( f(y) \) with \( n \) significant digits, we need only to

1. compute \( \varphi(n) \) in \( p_\varphi(n) \) steps,
2. compute \( d = \psi'(\varphi(n)) \) in \( O(\varphi(n)) \) steps, and
3. compute \( \psi(\varphi(n), d) \) in \( p_\psi(2 \varphi(n)) \) steps.

So, \( f \) is polynomial time computable. \( \square \)

We write \( f_n \xrightarrow{\text{poly}} f \) to denote that \( \{ f_n \} \) uniformly converges to \( f \) in polynomial bounded speed.

2.3. Roots

In the rest of this chapter we examine some basic theorems in
classical analysis and study them from the viewpoint of computational complexity.

A root of a recursive real function must be recursive \([9,17]\). However, as we will see in the next theorem, a root of a polynomial time computable function is not necessarily polynomial time computable. We assume that negative real numbers are represented by a negative sign followed by a positive binary number.

**Theorem 2.3.** There exists a polynomial time computable function \(f\) on \([0, 1]\) such that \(f\) is strictly increasing and \(f(0) < 0 < f(1)\) but \(f^{-1}(0)\) is not polynomial time computable.

**Proof.** Assume that \(x \in [0, 1]\) is not in \(PR\) and there is a function \(\varphi\) such that \(A(x, \varphi)\) and the time complexity for \(\varphi\) is, say, \(\lambda n[2^{n_0}n]\), for some constant \(n_0\). Also assume that \(x \in (3/8, 5/8)\). We will define a sequence \(\{f_n\}\) of recursively piecewise linear functions such that \(f_n \xrightarrow{\text{poly}} f\), for some \(f\).

The basic idea is to construct \(f\) such that, for a real number \(y\) which is close to \(x\) and a small integer \(n\), the machine \(M_f\) which computes \(f\) will temporarily assign the value \(0\) to \(M_f^Y(n)\). Only when the input integer \(n\) becomes so large that there is enough time to distinguish \(y\) from \(x\), \(M_f\) will assign a nonzero value to \(M_f^Y(n)\) and make \(f(y) \neq f(x)\).

First we inductively define two sequences \(\{d_k\}\) and \(\{\overline{d}_k\}\) of numbers in \(D\):

\[
d_1 = 0; \quad \overline{d}_1 = 1;
\]
\[ d_2 = \varphi(2) - 2^{-2}; \quad d_2^{-} = \varphi(2) + 2^{-2}; \]

and for \( k \geq 3, \)
\[
\begin{align*}
  d_k &= \max\{ d_{k-1}, \varphi(k) - 2^{-k} \}, \text{ and} \\
  d_k^- &= \min\{ d_{k-1}, \varphi(k) + 2^{-k} \}.
\end{align*}
\]

We observe the following facts:

1. For all \( k \geq 0, \ d_k, \ d_k^- \in D_k, \)
2. \( 0 = d_1 \leq d_2 \leq \ldots \leq d_k < x < d_k^- \leq \ldots \leq d_1 = 1, \)
3. In \( O(2^{(n_0+2)k}) \) steps, we can compute \( \{ d_i, d_i^- \}_{i=1}^k \).

Then, we define \( \{ f_n \} \) as following:

If \( 2^k < n < 2^{k+1} \), let \( f_n \equiv f_{2^k} \).

For \( n = 2^k, \ k > 0 \), let \( f_n \) be recursively piecewise linear

with \( \leq 2k \) break points:

\[ d_1, \ d_2, \ldots, \ d_k, \ d_k, \ldots, \ d_1, \]

such that for all \( y \in \{ d_j, \ d_j^- \}_{j=1}^k, \)

\[
f_n(y) = \begin{cases} 
-2^{-2^{i-1}}, & \text{if } i = \max\{ j \mid y = d_j \} \text{ and } i < k; \\
0, & \text{if } y = d_k \text{ or } d_k^-; \\
2^{-2^{i-1}}, & \text{if } i = \max\{ j \mid y = d_j^- \} \text{ and } i < k.
\end{cases}
\]

For instance, if \( 0 = d_1 < d_2 = d_3 < d_4 < d_3^- = d_3^- < d_2^- < d_1 = 1, \)

then \( f_{16} \) is piecewise linear and is determined by the following points: \( (d_1, \ -1/2), \ (d_3, \ -1/16), \ (d_4, \ 0), \ (d_4, \ 0), \ (d_2, \ 1/4), \)

and \( (d_1, \ 1/2). \)

It is clear that for any \( k, \ f_{2^k} \) and \( f_{2^{k+1}} \) differ only on

\( (d_i, d_{i+1}) \cup (d_{k+1}, d_j), \) if \( d_k < d_{k+1} < d_{k+1} < d_k \) and

\( i = \max\{ m \mid d_m < d_k \}, \ j = \min\{ m \mid d_m > d_k \}. \)
So, for all \( y \in [0, 1] \),

\[
\left| f_{2^k}(y) - f_{2^{k+1}}(y) \right| \\
\leq \max \left\{ \left| f_{2^k}(d_k) - f_{2^{k+1}}(d_k) \right|, \left| f_{2^k}(\overline{d}_k) - f_{2^{k+1}}(\overline{d}_k) \right| \right\} \\
\leq 2^{-(2^k)}
\]

and \( \{ f_n \} \) must uniformly converge to a function \( f \).

It is obvious that \( f \) is strictly increasing and \( f(x) = 0 \).

It is left to show that \( f \) is polynomial time computable. To see this, we observe:

1. In \( O(2^{(n_0+2)n}) \) steps, we can find \( \{ d_i, \overline{d}_i \}_{i=1}^k \);

2. In \( O(2^{n_1^k}) \) steps, for some \( n_1 \), we can find exact values of \( \left\{ f_{2^k}(d_i), f_{2^k}(\overline{d}_i) \right\}_{i=1}^k \) (by comparing \( d_i \)'s and \( \overline{d}_i \)'s and computing \( 2^{-(2^{i-1})} \) and \( 2^{-(2^{i-1})} \)).

So, for any \( d \), \( |d| = n = 2^k \), if \( \frac{d_k}{2^k} \leq d \leq \frac{\overline{d}_k}{2^k} \), then \( f_n(d) = 0 \); if \( d < \frac{d_k}{2^k} \) (or, \( d > \frac{\overline{d}_k}{2^k} \)), we can first compute \( \{ d_i, \overline{d}_i \}_{i=0}^k \) and find \( d_i \), \( d_{i+1} \) such that \( d_i \leq d \leq d_{i+1} \) (or, \( \overline{d}_i \), \( \overline{d}_{i+1} \) such that \( \overline{d}_i \leq d \leq \overline{d}_{i+1} \)) then by linear interpolation, find the exact value of \( f_n(d) \). The total time is

\[
\leq O(n_0^{n+2}n_1^{n+2})
\]

for some \( n_2 \geq 0 \). By corollary 2.2.1, \( f \) is polynomial time computable. \( \square \)

The above proof is based on the following idea. For \( y \neq x \), \( f(y) \) is chosen so close to \( 0 \) that when there is not enough time to distinguish \( y \) from \( x \), the value \( 0 \) is computed and when there
is time to distinguish \( y \) from \( x \), the exact value of \( f(y) \) is computed. This idea can be easily extended to \( x \) with higher time complexity.

**Corollary 2.3.1.** For any recursive \( x \) in \([0,1]\), there is a function \( g \in PF \) such that \( g(x) = 0 \).

**Proof.** Since \( x \) is recursive, there exists a recursive time function \( T \) such that, in \( T(n) \) steps, we can compute the first \( n \) significant digits of \( x \). I.e., in the proof of theorem 2.3, we can compute \( d_k \) and \( \overline{d}_k \) in \( O(\sum_{i=1}^{k} T(i)) \) steps. Let

\[
S(k) = \sum_{i=1}^{k} T(i),
\]

and let \( g_S(k) \) be piecewise linear with break points \( \{d_i, \overline{d}_i\}_{i=1}^{k} \) and

\[
ge_S(k)(d_i) = -2^{-S(i)}, \text{ if } d_i < d_{i+1} \leq d_k
\]
\[
ge_S(k)(\overline{d_i}) = 2^{-S(i)}, \text{ if } \overline{d_i} > \overline{d_{i+1}} \geq \overline{d_k}, \text{ and}
\]
\[
ge_S(k)(\overline{d_k}) = g_S(k)(\overline{d_k}) = 0.
\]

Also, \( g_n = g_S(k) \) if \( S(k) \leq n < S(k+1) \) (Note that \( T \) is the time complexity measure of \( x \) and hence "\( S(k) \leq n < S(k+1) \)" is a polynomial time predicate on \((n,k)\)).

Similar to the proof of the theorem, we can see that

\[
\text{poly}
\]

\[
g \overset{\text{def}}{=} g \text{ for some } g, \text{ and } |g_S(k) - g| < 2^{-S(k)}.
\]
So, by corollary 2.2.1, \( g \in PF \), and \( g(x) = 0 \).

Despite the fact that the roots of some functions are hard to compute, many numerical methods of finding roots exist. They are
useful only for special classes of functions, like polynomials, for example. The following theorem describes such a class.

**Definition 2.7.** A function \( f : [0, 1] \rightarrow \mathbb{R} \) is said to have a locally polynomially bounded modulus at \( x \) if there is a polynomial \( p \) such that for all \( n \in \mathbb{N} \) and \( y, z \in [0, 1] \),

\[
y \leq x \leq z \quad \text{and} \quad |y - z| < 2^{-p(n)} \implies |f(y) - f(z)| < 2^{-n}.
\]

**Theorem 2.4.** Let \( f \) be in \( PF \) and strictly increasing on \([0,1]\). If \( y \in \text{Range } f \) and \( y \in \text{PR} \) and \( f^{-1} \) has a locally polynomially bounded modulus at \( y \), then \( f^{-1}(y) \in \text{PR} \).

**Proof.** Assume that \( M \) computes \( f \) with time function \( p_M \), a polynomial. Also assume that \( q_f \) is a polynomial bound of modulus of \( f \) on \([0,1]\) and \( p \) is a polynomial bound of modulus of \( f^{-1} \) at \( y \).

Without loss of generality, let \( y = 0 \).

We now give an algorithm of computing \( f^{-1}(y) \).

For input \( k \), we proceed in stages.

Prior to stage 0, let \( a_0 = 0 \), \( b_0 = 1 \), and \( c_0 = 1/2 \).

**Stage \( n \).** Get \( d_n = M^{c_n}(p(k)+1) \), and compare it with \( 2^{-p(k)} \) and \( 2^p(k) \).

**Case n.1.** \( d_n > 2^{-p(k)} \).

Let \( a_{n+1} = a_n \), \( b_{n+1} = c_n \), \( c_{n+1} = \frac{a_{n+1} + b_{n+1}}{2} \)

and go to the next stage.

**Case n.2.** \( d_n \leq 2^{-p(k)} \).
Let \( a_{n+1} = c_n, \ b_{n+1} = b_n, \ c_{n+1} = \frac{a_{n+1} + b_{n+1}}{2} \)

and go to the next stage.

Case n.3. \( -2^{-p(k)} < d_n < 2^{-p(k)} \).

Halt and output \( c_n \).

End of algorithm.

First observe that if this algorithm halts at some stage \( n \),
then \( |f^{-1}(0) - c_n| < 2^{-k} \), since, from case n.3, we have
\[ |f(c_n) - 0| < 2^{-p(k)}. \]

It is left to show that the algorithm always halts in polynomial time.

We know that for any \( x_1, x_2 \in [0, 1] \),
\[ |x_1 - x_2| < 2^{-q_f(p(k)+1)} \implies |f(x_1) - f(x_2)| < 2^{-(p(k)+1)}. \]

Also, at each stage \( n \), if the algorithm does not halt, it is
always the case that \( |a_{n+1} - b_{n+1}| = |a_n - b_n|/2 \). So, by stage
\( m = q_f(p(k)+1) \),
\[ |a_m - b_m| = 2^{-q_f(p(k)+1)}. \]
and hence
\[ |f(c_m)| < 2^{-(p(k)+1)}. \]

since we can easily see that \( f(a_n) \leq y \) and \( f(b_n) \geq y \) for all \( n \leq n \).

That is, the algorithm must halt by stage \( m = q_f(p(k)+1) \). During
any stage before halting, \( p_M(p(k)+1) \) steps are required to find
\( d_n \). So the algorithm as a whole halts in
\( O(q_f(p(k)+1) \cdot p_M(p(k)+1)) \) steps. \( \square \)
Corollary 2.4.1. Let \( f \in \mathbb{P} \) be one-to-one on \([0,1]\). Then \( f^{-1} \in \mathbb{P} \) if and only if \( f^{-1} \) has a polynomially bounded modulus on \( \text{Range } f \).

Theorem 2.4 can be immediately applied to some interesting classes of functions.

Theorem 2.5. If \( f \) is analytic on \([0,1]\) (i.e. \( f \in \mathcal{C}^\infty[0,1] \)), and the power series of \( f \) at \( 1/2 \) uniformly converges to \( f \) on \([0,1]\) and is strictly increasing, then \( f^{-1} \) has a polynomially bounded modulus on \([0,1]\).

Proof. We use \( f \in \mathbb{P}[0,1] \) to indicate that \( f \) has a polynomially bounded modulus on \([0,1]\).

Three lemmas are first established.

Lemma 2.5.1. (i) If \( f \in \mathbb{P}[a,b] \), and \( f \in \mathbb{P}[b,c] \), then \( f \in \mathbb{P}[a,c] \).

(ii) If \( f \in \mathbb{P}[a,b] \), then \( f_1(x) = f(x-c) \) is in \( \mathbb{P}[a+c,b+c] \), and \( f_2(x) = f(x) + c \) and \( f_3(x) = c \cdot f(x) \) are in \( \mathbb{P}[a,b] \).

Proof. Obvious.

Lemma 2.5.2. If \( f \in \mathcal{C}^3[0,1] \), and \( f' > 0 \) on \([0,1]\), then \( f^{-1} \in \mathbb{P}[0,1] \).

Proof. By the mean-value theorem,

\[
|f(x) - f(y)| > \min_{t \in [0,1]} f'(t) \cdot |x - y|
\]

for all \( x, y \in [0,1] \).
Lemma 2.5.3. If, in addition to the assumption in the theorem, $f'$ has exactly one zero in $[0,1]$, then $f^{-1} \in PM [0,1]$.

Proof. Without loss of generality, let $f'(1/2) = 0$.

Let $f(x) = \sum_{n=0}^{\infty} a_n (x-1/2)^n$ and let $a_k$ be the first nonzero coefficient (except $a_0$).

First claim that $(\exists n_1) (\forall n \geq n_1)$

$$|a_k|/2 \geq \sum_{i=k+1}^{\infty} |a_i| \cdot 2^{-(i-k)n}$$

(*)

(proof of claim) Since the power series converges on $[0,1]$, the radius of convergence $\geq 1/2$, and so

$$\limsup_{n \to \infty} |a_n|^{1/n} \leq 2$$

and $(\exists n_0) (\forall n \geq n_0)$ $|a_n| < 2^{2n}$.

Let $n_1$ be so large that

(1) for $j = 1, 2, \ldots, n_0 - k,$

$$2^{-j \cdot n_1} \cdot |a_{k+j}| < 2^{-(j+1)} \cdot |a_k|,$$

and

(2) $2^{-(n_0-k) n_1} \cdot 2^{2n_0} = 2^{-(n_0+1)} \cdot |a_k|,$

then

$$\sum_{i=k+1}^{n_0} |a_i| \cdot 2^{-(i-k)n}$$

$$\leq \sum_{i=k+1}^{n_0} |a_i| \cdot 2^{-(i-k)n} + \sum_{i=n_0+1}^{\infty} |a_i| \cdot 2^{-(i-k)n}$$

$$< \sum_{i=k+1}^{n_0} 2^{-(i-k+1)} \cdot |a_k| + \sum_{i=n_0+1}^{\infty} 2^{2i} \cdot 2^{-(i-k)n}$$

$$< |a_k|/2$$

for all $n \geq n_1$. (end of the proof).

Now, we choose $n_2$ so that $n_2 \geq n_1$ and the only possible root for $f''$ in $[1/2 - 2^{-n_2}, 1/2 + 2^{-n_2}]$ is $1/2$ (this is
possible because $f$ is analytic and $f''$ can have only isolated roots. Then, from lemma 2, $f^{-1} \in \text{PM} \left[0, 1/2 - \frac{n}{2}\right]$ and $f^{-1} \in \text{PM} \left[1/2 + 2^{-n}, 1\right]$.

On $\left[1/2 - \frac{n}{2}, 1/2 + 2^{-n}\right]$, if $|x - y| \geq 2^{-n}$, then there are two cases.

Case 1. $x \leq 1/2 < y \ (\text{or } y \leq 1/2 < x)$.

In this case, $|x - 1/2| \geq 2^{-(n+1)}$.

Thus, $|f(x) - f(1/2)| \geq \sum_{i=k}^{\infty} a_i \cdot 2^{-(n+1)i}$

$\geq 2^{-(n+1)k} \left(|a_k| - \sum_{i=k+1}^{\infty} |a_i| \cdot 2^{-(n+1)(i-k)}\right)$

$\geq 2^{-(n+1)k} \cdot (|a_k|/2)$

So, $|f(x) - f(y)| \geq 2^{-(n+1)k} \cdot (|a_k|/2)$.

Case 2. $1/2 < x < y \ (\text{or } x < y < 1/2)$.

Since $f' = 0$ at $1/2$, $f' > 0$ on $\left(1/2, 1/2 + 2^{-n}\right)$ and $f'' \neq 0$ on $\left(1/2, 1/2 + 2^{-n}\right)$, we have in this case $f'' > 0$ on $\left(1/2, 1/2 + 2^{-n}\right)$. Thus $f$ is concave upward on $\left(1/2, 1/2 + 2^{-n}\right)$, and hence for $|x - y| \geq 2^{-n}$

$|f(x) - f(y)| \geq |f(1/2 + 2^{-n}) - f(1/2)|$

$\geq 2^{-(n+1)k} \cdot (|a_k|/2)$.

In summary, $f^{-1} \in \text{PM} \left[1/2 - 2^{-n}, 1/2 + 2^{-n}\right]$, and $f^{-1} \in \text{PM} \left[0, 1\right]$ follows. (end of the proof of lemma 2.5.3)

(continuation of the proof of theorem) Since $f$ is analytic, we can have only a finite number of zeroes of $f'$ on $[0,1]$ and by lemma 1 and lemma 3, $f^{-1} \in \text{PM}[0,1]$. []
Lemma 2.1. If \( f \in C^2[0,1] \) (i.e., \( f'' \) exists and is continuous) and \( f \) is in PF then \( f' \in \text{PF} \) on \([0,1]\).

Proof. Since \( f'' \) is continuous on \([0,1]\), it is bounded above, say, by \( 2^M \). First, assume that \( x \in [2^{-M}, 1-2^{-M}] \).

For any \( \varphi, A(x, \varphi) \), and input \( k \), we compute \( \varphi(M+k+2) \), and let

\[
\begin{align*}
\bar{x} &= \varphi(M+k+2) - 2^{-(M+k+2)} \\
\overline{x} &= \varphi(M+k+2) + 2^{-(M+k+2)}.
\end{align*}
\]

Then compute \( d = (M_x(M+2k+3) - M_{\overline{x}}(M+2k+3)) \cdot 2^{M+k+1} \) and output it. Then we have

\[
|f'(x) - d| \
\leq |f'(x) - (f(\bar{x}) - f(x)) \cdot 2^{M+k+1}| \\
+ 2^{M+k+1} |(f(\overline{x}) - f(\bar{x})) - (M_x(M+2k+3) - M_{\overline{x}}(M+2k+3))|
\]

\[
\leq |f'(x) - f'(x_1)| + 2^{-(k+1)}, \quad \text{for some} \ x_1 \in (x, \overline{x})
\]

(by the mean-value theorem)

\[
= |x - x_1| \cdot f''(x_2) + 2^{-(k+1)}, \quad \text{for some} \ x_2 \in (x, x_1)
\]

\[
< 2^{-(k+2)} + 2^{-(k+1)}
\]

\[
< 2^{-k}.
\]

Now, for \( x \in [0, 2^{-M}] \) (or, \( x \in [1-2^{-M}, 1] \)), choose

\[
\bar{x} = \varphi(M+k+2) \quad \text{and} \\
\overline{x} = \varphi(M+k+2) + 2^{-(M+k+1)}
\]

and output \( d \).

Then \( (f(\bar{x}) - f(x)) \cdot 2^{M+k+1} = f'(x_1) \) for some \( x_1 \) in \((\bar{x}, \overline{x})\).

Since \( |x - x_1| < 2^{-(M+k)} \), we have \( |f'(x) - d| < 2^{-(k-1)}. \)
Corollary 2.5.1. All roots of an analytic, polynomial time computable function are polynomial time computable.

Proof. Let $f$ be analytic and $f \in \mathcal{P}\mathcal{F}$ on $[0,1]$. If $x$ is a root of $f$ of order $1$, then there is an interval $[a,b]$ such that $x \in [a,b]$, and $f$ is strictly increasing or strictly decreasing on $[a,b]$. By theorem 2.4 and theorem 2.5, we have $x \in \mathcal{P}\mathcal{R}$. If $x$ is a root of $f$ of order $m > 1$, then $x$ is a root of $f^{(m-1)}$ of order $1$. $f^{(m-1)}$ is analytic and by lemma 2.1, it is in $\mathcal{P}\mathcal{F}$ also. So, $x \in \mathcal{P}\mathcal{R}$. []

Corollary 2.5.2. $\mathcal{P}\mathcal{R} [1]$ is an algebraically closed field.

Proof. For $a, b \in \mathcal{P}\mathcal{R}$, $a \neq 0$, the following four polynomials are polynomial time computable and their roots are $-a$, $a+b$, $1/a$, and $a \cdot b$, respectively.

$$x + a; \quad x - a; \quad a \cdot x = 1; \quad x/a - b.$$

So, by corollary 2.5.1, $\mathcal{P}\mathcal{R}$ is a field, and hence a real closed field. []

Theorem 2.5 can not be extended even to $C^\infty[0,1]$. Let $f$ be defined on $[-1,1]$ as follows:

$$f(x) = \begin{cases} e^{-1/x^2}, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ e^{-1/x^2}, & \text{if } x < 0 \end{cases}$$

Then, $f$ is in $C^\infty[0,1]$ and is strictly increasing. Since $f^{(n)}(0) = 0$, for all $n \geq 0$, $f$ is not analytic.
We claim that $f$ is polynomial time computable.

(Proof). Since $x^2$, $1/x$, and $e^x$ are polynomial time computable, the only task is to distinguish the given real number $y$ from 0. However, for $n$ large enough, $|y| < 2^{-n} \implies |f(y)| < 2^{-n}$. Hence, for any $y$, if $y$ is close to 0, then output 0; otherwise, compute $e^{-1/y^2}$ or $-e^{-1/y^2}$. 

It is obvious that $f^{-1}$ does not have a locally polynomially bounded modulus at 0, because $f(2^{-n}) = e^{-2^{2n}}$, for all $n \geq 0$.

So, $f \in C^\infty[-1,1]$, strictly increasing, and is polynomial time computable, but $f^{-1}$ does not have a locally polynomially bounded modulus at 0. This result supports the next conjecture.

Conjecture. There exists a function $f$ in $C^\infty[0,1]$ such that $f$ is polynomial time computable but it has a root which is not in PR.

One of the most interesting and important task in numerical analysis is to find a uniform method of computing the roots of polynomial equations $[5,16,26]$. First we need to warn that a "uniform" method is not to be confused with a "continuous" method. By a uniform method of finding roots of polynomials, we mean an algorithm which, given an input $n$ and $n+1$ complex numbers (in oracle form) $a_0$, $a_1$, ..., $a_n$ (or, $2n+2$ real numbers), computes $n$ complex numbers $x_1$, $x_2$, ..., $x_n$ which are the only roots of the polynomial $\sum_{j=0}^{n} a_j z^j$. By a continuous method, we mean that the above described algorithm, in addition to being uniform, actually computes
a function $F : \bigcup_{n=0}^{\infty} \{ n \} \times \mathbb{C}^{n+1} \to \mathbb{C}^n$ such that

(i) for all $n = 0, 1, 2, \ldots$, and $<a_0, a_1, \ldots, a_n> \in \mathbb{C}^{n+1}$, $F(n; a_0, \ldots, a_n)$ is a sequence of the $n$ complex roots of the polynomial $\sum_{j=0}^{n} a_j z^j$, and

(ii) for all $n = 0, 1, 2, \ldots$, the function $F_n : \mathbb{C}^{n+1} \to \mathbb{C}^n$ defined by $F_n(a_0, \ldots, a_n) = F(n; a_0, a_1, \ldots, a_n)$ is continuous on $\mathbb{C}^{n+1}$.

From the theory of complex variables, there does not exist a continuous function $G : \mathbb{C} \to \mathbb{C}^2$ such that $G(z) = (z_1, z_2)$ and $z_1$ and $z_2$ are the two square roots of $z$. So, the above described function $F$ does not exist, and all we can expect is a polynomial time uniform algorithm.

Recently methods of finding the roots of polynomials using Sturm sequences have been studied [5,16,26] and several algorithms have been proposed. Among them, Pinkert [16] demonstrated a fast algorithm which works for all polynomials with rational complex coefficients. Another fast algorithm on all complex polynomials based on the principle of argument and fast evaluation of the Sturm sequences is described by Wilf [26]. It seems that we can modify the "termination procedure" of Wilf's algorithm so that it becomes a polynomial time algorithm for finding roots of polynomials.

We need to verify the following items:

1. In Wilf's algorithm, when the number of zeros in a rectangle $R$ is computed by observing the number of sign variations in the
Sturm sequences evaluated at each side of the rectangle, Wilf suggested to check the evenness of the sign variations to make sure of no loss of significance. We feel that although it is practically useful it may not be sufficient. In order to guarantee the significance of the computation, we need to show

Lemma A. For a polynomial $f$ and an integer $n$, there is a polynomial $p$ such that if no roots $x$ of $f$ are close to the boundary $b(R)$ of $R$ so that $d(x, b(R)) > 2^{-p(n)}$ then the above mentioned technique of computing the number of roots of $f$ in $R$ is correct.

Conversely, if there is at least one root $x$ of $f$ are close to $b(R)$ such that $d(x, b(R)) < 2^{-p(n)}$ then there is a polynomial time checking algorithm (say, the evenness checking suggested in [26]) to point out this fact.

2. When a loss of significance is discovered (by the algorithm in Lemma A) that some roots are close to the boundary of the rectangle $R$, instead of trying at most three different partitions of the bigger rectangle as suggested by [26] we can try $N+1$ ($N$ = degree of $f$) times, each time moving the boundary of the new rectangle $2^{-(p(n)+1)}$ from the boundaries of the previous failed rectangles so that no root $x$ can be close to any two rectangles. Since $f$ has only $N$ roots, at least one of the $N+1$ partitions will provide a correct evaluation of the number of roots in rectangles. Note that each partition still separates the old rectangles into four subrectangles of "almost" equal size.
Thus we have the following conjecture:

**Conjecture.** There exists a uniform algorithm of finding all the roots of polynomials with its time complexity bounded by \( p(n,N) \), a polynomial over two variables \( n \), the number of significant digits of the roots, and \( N \), the degree of the polynomial whose roots are to be found.

If this conjecture is true, then a further interesting question is whether there is an extension of this algorithm to larger classes of functions, say, the class of analytic functions with polynomial time computable power series. It seems that we need some powerful
technique similar to Sturm sequences to apply the principle of argument to analytic functions.

2.4. Derivatives

In this section, we try to answer these questions about derivatives of polynomial time computable real functions: Is a polynomial time computable function differentiable? If it is differentiable, is its derivative polynomial time computable?

First, we show that a polynomial time computable function is not necessarily differentiable.

**Theorem 2.6.** There exists a function $f \in PF$ which is nowhere differentiable on $[0,1]$.

**Proof.** Define

$$\phi(x) = \begin{cases} x & \text{if } 0 \leq x \leq 1 \\ 2-x & \text{if } 1 \leq x \leq 2 \end{cases}$$

and extend the definition of $\phi(x)$ to all real $x$ by requiring that $\phi(x+2) = \phi(x)$.

Define

$$f(x) = \sum_{n=0}^{\infty} \phi(4^n x), \quad 0 \leq x \leq 1.$$ 

$f$ is nowhere differentiable [21]. It is also provably polynomial time computable.

Since

$$\sum_{n=3k+7}^{\infty} (3/4)^n \phi(4^n x) \leq \sum_{n=3k+7}^{\infty} (3/4)^n \leq (1/2)^k,$$

we only need to show that in $p_1(k)$ steps for some polynomial $p_1$, we can compute a number $d$ such that
\[ d = \sum_{n=0}^{3k+6} \phi (4^n x) \leq 2^{-k}. \]

Since \( \phi \) is piecewise linear and polynomial time computable, we assume that in order to compute \( k \) significant digits of \( \phi(y) \), it takes only \( p_2(k) \) steps, where \( p_2 \) is a polynomial. Now, \( d \) can be computed by the following procedure:

1. For each of \( n = 0, 1, \ldots, 3k+6 \), get \( d_n \) such that
   \[ |d_n - 4^n x| < 2^{-p_2(k+4)}; \]
2. For each of \( n = 0, 1, \ldots, 3k+6 \), get \( e_n \) such that
   \[ |e_n - \phi(d_n)| < 2^{-p_2(k+4)}; \]
3. For each of \( n = 0, 1, \ldots, 3k+6 \), get \( e_n' \) such that
   \[ |e_n' - e_n (3/4)^n| < 2^{-(k+3+n)}; \]
4. Add \( e_n' \), \( n = 0, 1, \ldots, 3k+6 \), to get \( d \).

Then,
\[ d = \sum_{n=0}^{3k+6} (3/4)^n \phi (4^n x) \]
\[ \leq \sum_{n=0}^{3k+6} |e_n' - (3/4)^n \phi (4^n x)| \]
\[ \leq \sum_{n=0}^{3k+6} (|e_n' - e_n (3/4)^n| + (3/4)^n |e_n - \phi (4^n x)|) \]
\[ \leq \sum_{n=0}^{3k+6} (2^{-(k+3+n)} + (3/4)^n (|e_n - \phi(d_n)| + |\phi(d_n) - \phi(4^n x)|)) \]
\[ \leq 2^{-(k+2)} + 4 \left( 2^{-(k+4)} + 2^{-(k+4)} \right) \]
\[ = 2^{-k} \]

Also, the computation time is bounded by \( O( k^2 p_2(k+4) ) \).

So, \( f \) is polynomial time computable. \( \square \)
Whether a derivative of a polynomial time computable function must be polynomial time computable or not is not known. We do know that the modulus of continuity of a derivative determines its complexity.

Theorem 2.7. If \( f \in PF \) and \( f' \) exists and is continuous on \([0,1]\), then \( f' \in PF \) if and only if \( f' \) has a polynomially bounded modulus on \([0,1]\).

**Proof.** The "only if" part follows from theorem 2.1.

For the "if" part assume that \( p \) is a polynomial bound of modulus of \( f' \), i.e., for any \( y, z \in [0,1] \),

\[
|y - z| < 2^{-p(n)} \implies |f'(y) - f'(z)| < 2^{-n}.
\]

Also assume that the OTM \( M \) computes \( f \) in polynomial time.

We compute \( f' \) by the following algorithm.

**Step 1.** For given \( \varphi, A(x, \varphi) \), and input \( n \), we compute \( \varphi(p(n+2)) \) and \( \overline{x} \) and \( \underline{x} \):

- **Case 1.** \( 0 < \varphi(p(n+2)) < 1 \).
  - Let \( \underline{x} = \varphi(p(n+2)) - 2^{-p(n+2)} \),
  - \( \overline{x} = \varphi(p(n+2)) + 2^{-p(n+2)} \), and go to step 2.

- **Case 2.** \( \varphi(p(n+2)) = 0 \).
  - Let \( \underline{x} = 0, \overline{x} = 2^{-p(n+1)} \), and go to step 2.

- **Case 3.** \( \varphi(p(n+2)) = 1 \).
  - Let \( \underline{x} = 1 - 2^{-p(n+1)}, \overline{x} = 1 \).

**Step 2.** Then, output \( (M^\overline{x}(p(n+2)+n+2) - M^\underline{x}(p(n+2)+n+2))2^{p(n+1)} \).

End of algorithm.
For any case, \( x \in (\overline{x}, \underline{x}) \), and

\[
\left| f''(x) - \left( \frac{p}{n+2} + \frac{n+2}{n+2} \right) \cdot 2^{p(n+1)} \right| \\
\leq \left| f''(x) - \left( f(\overline{x}) - f(\underline{x}) \right) \cdot 2^{p(n+1)} \right| \\
+ 2^{p(n+1)} \left( \left| f(\overline{x}) - \frac{x}{n+2} \right| + \left| f(\underline{x}) - \frac{x}{n+2} \right| \right) \\
\leq \left| f''(x) - f''(\overline{x}) \right| + 2^{p(n+1)} \cdot 2^{-(p(n+2)+n+1)}, \text{ for some } \overline{x} \in (\underline{x}, \overline{x}) \\
(\text{by the mean-value theorem}) \\
< 2^{-(n+1)} + 2^{-(n+1)} = 2^{-n} \\
\]

So, lemma 2.1 is actually a corollary of this theorem since the fact that \( f'' \) exists and is continuous implies that \( f' \) has a polynomially bounded modulus.

It also follows that if an analytic function is polynomial time computable, then every coefficient of its power series is polynomial time computable. For most cases, we do use the power series to compute the function values.

**Definition 2.3.** A sequence \( \{x_n\} \) of real numbers is said to be polynomial time computable if there exists a TM \( M \) and a polynomial \( p \) such that

(i) for any \( n \in \mathbb{N} \), the function \( \psi_n = \lambda k[M(n,k)] \) computes \( x_n \), i.e., \( A(x, \psi_n) \), and

(ii) for any \( n \) and \( k \in \mathbb{N} \), \( M(n,k) \) is computable in \( \leq p(n+k) \) steps.

It is easy to see that the idea of using a power series of a function to compute its function values requires the sequence of
coefficients of the power series to be polynomial time computable. Unfortunately, although we know that every coefficient is polynomial time computable, we do not know the complexity of the sequence of coefficients.

**Open question.** If \( f \in PF \) is analytic on \([-1,1] \), is the sequence \( \{ f^{(n)}(0) \}_{n=0}^{\infty} \) polynomial time computable?

Note that a negative answer to this question will exhibit an example of a polynomial time computable analytic function whose power series does not provide an efficient computation.

### 2.5. Integrals

Since all polynomial time computable functions are continuous, they are also Riemann integrable. In addition, the integral function \( I(x) = \int_{0}^{x} f \) of a polynomial time computable function \( f \) has a polynomially bounded modulus, because for any \( x, y \) in \([0,1] \),

\[
|I(x) - I(y)| = \left| \int_{x}^{y} f \right| < \max_{0 \leq t \leq 1} f(t) \cdot |x - y|.
\]

However, the exact complexity of integrals of polynomial time computable functions is not known. In the following we consider the complexity of a uniform method of integration and the classification of the complexity of the integration problem into nondeterministic classes.

By a uniform method, we mean a functional which maps functions to real numbers. A functional \( F \) is said to be computable if there exists a two-oracle TM \( M_F \) with the following interpretations:
1. The first oracle $m$ is a modulus oracle. It answers the queries like "if we want $n$ significant digits of $f(x)$ to be output from the second oracle $f$, how many significant digits of $x$ do we need to supply?"

2. The second oracle $f$ is a function oracle, and it answers the queries like "if $y$ is a number such that $|y - x| < 2^{-m(n)}$, please give me a number $z$ such that $|z - f(x)| < 2^{-n}$". such that $M_f$ computes a function $\varphi$ and $A(f(x), \varphi)$. If, for any function $f$ and its modulus $m$, the computation time of $M_f$ is bounded by a polynomial $p$, then we say that $f$ is polynomial time computable.

Now we show that

1. the functional $I(f) = \int_0^1 f$ is not polynomial time computable, and

2. if $f \in PF \Rightarrow \int_0^1 f \in PR$ then $P \neq PSPACE$ (for discussion of "$P = SPACE$?" problem, see [2]).

Theorem 2.8. The functional $I(f) = \int_0^1 f$ on $PF$ is not polynomial time computable.

Proof. By way of contradiction, assume that $I$ is polynomial time computable and, say, that the time complexity is bounded by $p$, a polynomial. Then, for input $k$, it can query the function oracle $f$ at most $p(k)$ times. However, it is not hard to show that for any large enough $k$ and any choice of $p(k)$ points $x_1 < \ldots < x_p(k)$ in $D[0,1]$, there exist two functions $f$ and $g$ on $[0,1]$ such
that

(i) both \( f \) and \( g \) are in PF and \( m_f \) and \( m_g \), their modulus functions are exactly the same,

(ii) \( f(x_i) = g(x_i) \), for all \( i = 1, \ldots, p(k) \), and

(iii) \( \left| \int_0^1 f - \int_0^1 g \right| \geq 2^{-k} \).

For example, let \( f \equiv 0 \) and \( g \) be a piecewise linear function determined by the following points:

\[
0 = x_0 < x_1 < x_2 < \ldots < x_{p(k) + 1} = 1
\]

and \( g(x_i) = 0 \), for all \( i = 0, 1, \ldots, p(k) + 1 \),

\[
g\left( \frac{x_i + x_{i+1}}{2} \right) = \frac{x_{i+1} - x_i}{2}, \text{ for all } i = 0, 1, \ldots, p(k),
\]

and let \( m_f = m_g = \text{identity function} \).

Now, exactly the same computation will be performed for the oracles \((m_f, f)\) or the oracles \((m_g, g)\) but then at least one of the outputs has error \( 2^{-k} \) and leads to a contradiction. Thus, we have shown that \( I \) is not polynomial time computable. \( \square \)

**Definition 2.9.** A real number \( x \) is said to be in PRSPACE, if there exists a TM \( M \) which computes a function \( \varphi \) such that \( A(x, \varphi) \) and the number of cells used by \( M \) on \( n \) is \( \leq p(n) \) for some polynomial \( p \).

**Theorem 2.9.** If \( f \in \text{PF} \) then \( \int_0^1 f \in \text{PRSPACE} \).

**Proof.** Let \( p \) be the time complexity function of the TM \( M \) which computes \( f \). Consider the following PL/I-like algorithm for \( \int_0^1 f \):
\[ i = 0 ; \]
\[ \text{do } d = 0 \text{ to } 1 \text{ by } 2^{-p(k)} ; \]
\[ i = i + M^d(k) * 2^{-p(k)} ; \]
\[ \text{end ;} \]

The computation of \( M^d(k) \) can be done in \( p(k) \) cells. Totally we only need \( 4 \cdot p(k) + O(k) \) cells (\( p(k) \) cells to compute \( M^d(k) \), \( 2 \cdot p(k) \) more cells to store \( M^d(k) \cdot 2^{-p(k)} \) and \( i \), and \( p(k) + O(k) \) cells for updating \( d \) ). Hence, \( \int_0^1 f \in \text{PRSPACE} \). \[ \square \]

**Corollary 2.9.1.** If \( f \in \text{FP} \), then \( \int_0^1 f \in \text{EXPR} \).

It is clear that \( \text{PR} \neq \text{PRSPACE} \Rightarrow \text{P} \neq \text{PSPACE} \). So, although we believe that some \( f \in \text{FP} \) has integral \( \int_0^1 f \notin \text{PR} \), it is hard to prove (if it is possible at all). But we may locate the complexity of integrals more accurately if we may show the following:

**Conjecture.** There exists a function \( f \in \text{FP} \) and a \( \text{PSPACE} \)-complete function \( \varphi \) such that \( A(\int_0^1 f, \varphi) \).

If we put more restrictions on the functions some integration techniques may be polynomial time computable; however, even for analytic functions, we do not know any uniform techniques. The straightforward method of integrating the power series won't work because we do not know the polynomial time convergence of the power series. Other known numerical integration techniques such
as composite Simpson's rule and quadrature formulas are not polynomial time computable although they work efficiently for a small number of significant digits.

2.6. Maximum Values

It is shown in [9,24] that the maximum value of a recursive real function is recursive. As far as polynomial time computable functions are concerned, the complexity of the maximum values of polynomial time computable functions is, similar to that of integration, not known. We can show that

1. the functional \( \text{MAX}(f) = \max_{0 \leq t \leq 1} f(t) \) is not polynomial time computable, and

2. if \( f \in PF \Rightarrow \max f \in PR \), then \( P \neq NP \) (for discussions of "P = NP?" problem, see [2]).

Theorem 2.10. The functional \( \text{MAX} \) on \( PF \) is not polynomial time computable.

Proof. In the proof of theorem 2.8, we showed the existence of two functions \( f \) and \( g \) both being in \( PF \) and for large enough \( n \), in polynomial time, we cannot distinguish them but

\[ |\max f - \max g| \geq 2^{-n} \].

So, \( \text{MAX} \) is not polynomial time computable. \( \Box \)

Theorem 2.11. If \( f \in PF \), then \( \max f \in PR^{NP} \). i.e., \( \max f \) can be computable in polynomial time using some \( NP \) set as an oracle.

Proof. For any \( f \in PF \), let the OTM \( M_f \) compute \( f \) with time
complexity bounded by \( p_f \), a polynomial.

Let \( U = \{ d \in D \mid (\exists x) M^x_f(|d|) \geq d \} \). Then,

\[
\{ d \in U \mid (\exists x) M^x_f(|d|) \geq d \} \\
\iff (\exists e) \{ e \leq p_f(|d|) \mid M^e_f(d) \geq d \}
\]

Since \( M^e_f(|d|) \geq d \) is a polynomial time relation on \( (d,e) \), \( U \in \text{NP} \).

Now we use \( U \) as an oracle to compute \( \max f \):

Given \( k \in \mathbb{N} \), first compute \( p_f(1) \), and simulate \( M^d_f(1) \) for all \( d \in \{ \lfloor \frac{p_f(l)}{2} \rfloor \mid 0 \leq l \leq \lfloor \frac{p_f(1)}{2} \rfloor \} = D_{p_f}(1) \) and determine \( m_o = \max \{ M^d_f(1) \mid d \in D_{p_f}(1) \} \). Then \( m_o - l < \max f < m_o + l \) and the above computation only takes a constant number of steps.

Now, binary search \( e \in [m_o - 1, m_o + 1] \cap D_k \) such that

\[
(\exists d) \mid d \mid \leq p_f(k) \mid e \leq M^d_f(k) \ \text{and} \\
(\forall d) \mid d \mid \leq p_f(k) \mid e + 2^{-k} > M^d_f(k)
\]

by querying \( U \) \( k + 1 \) times and using only \( O(k) \) steps.

Then,

\[
|e - \max f| \\
\leq |e - \max \{ M^d_f(k) \mid d \leq p_f(k) \}| \\
+ |\max f - \max \{ M^d_f(k) \mid d \leq p_f(k) \}| \\
< 2^{-k} + 2^{-k} \\
= 2^{-(k+1)}
\]
Corollary 2.11.1. If \( f \in PF \), then \( \max f \in \text{TIME}_R(2^{\text{Poly}}) \).

Proof. The queries of membership in \( U \) can be answered in exponential time (see [2] for more discussions on the relationship between exponential time and nondeterministic polynomial time computation).

Corollary 2.11.2. If \( P = \text{NP} \), then \( f \in PF \Rightarrow \max f \in PR \).

Conjecture. If \( P \neq \text{NP} \), then there exists a function \( f \in PF \) such that \( \max f \notin PR \).

One approach to the above conjecture and the one about integral (that there exists \( f \in PF \) such that \( \int_0^1 f \) is \( \text{PSPACE} \)-complete) is to find some known \( \text{NP} \)-complete (or, \( \text{PSPACE} \)-complete) problem with the similar nature and reduce it to the maximum value problem (or integral problem). We failed to do so because very few known \( \text{NP} \)-hard problems can be the candidates to be reduced to these problems with build-in "continuous" property. Under such circumstances, it may still be helpful to understand more about the complexity of these "continuous" procedures by deriving relationships among them, e.g., by showing that the maximum value problem is polynomial time reducible to the integral problem.
Chapter III
Conclusion

In this part we discussed the computability and the complexity problems of real functions. We have introduced some basic and natural approaches. In this chapter we discuss the open problems and point out some alternative approaches.

One of the most basic concepts which has been widely accepted is that only continuous functions are computable. We have shown that this results holds under our oracle-machine model. In addition, we have shown the existence of computable partial functions so that the step functions with recursive jump points are computable if we ignore the jump points. However, we only discussed the complexity of functions continuous on \([0,1]\). We would like to know the complexity of the step functions too. It is quite reasonable to guess that the function

$$ g_{x_0}(x) = \begin{cases} 1, & \text{if } 0 \leq x < x_0, \\ 0, & \text{if } x_0 < x \leq 1, \\ \uparrow, & \text{if } x = x_0 \end{cases} $$

is polynomial time computable if \(x_0\) is in PR. However, under our standard definition of time complexity, we know that there is no polynomial time bound within which we can distinguish \(x\) from \(x_0\) even if \(x_0\) is some "easy" number like 1/2 (see section 2.1).
If we look at the computation of $g_{1/2}$ carefully we will see that for most of numbers $x$, we can quickly get an answer to the question whether $x < 1/2$ or not, and only for a small portion of numbers in $[0,1]$, we may need a large amount of time to answer this question. This suggests the following definition.

Let $m$ be a function defined on all open sets in $[0,1]$ such that if $S$ is the disjoint union of open intervals $\{(a_i, b_i)\}_{i=1}^\infty$, then $m(S) = \sum_{i=1}^\infty (b_i - a_i)$. That is, $m$ is the Lebesgue measure restricted to the open sets contained in $[0,1]$.

**Definition 3.1.** Let $f : S \rightarrow R$, $S \subseteq [0,1]$, be a partial recursive function. We say that $f$ is polynomial time approximable if there are an OTM $M$, a sequence $S_n$ of open sets with rational endpoints and a polynomial $p$ such that

(i) $(\forall n) [S_n \subseteq S_{n+1} \subseteq S]$

(ii) $(\forall n) [m(S) - m(S_n) \leq 2^{-n}]$

(iii) $(\forall n) (\forall x \in S_n) (\forall \varphi) [A(x, \varphi) \Rightarrow (\forall k \in n) [M^\varphi(k) \text{ converges in } \leq p(k) \text{ steps and } |M^\varphi(k) - f(x)| < 2^{-k}]].$

Intuitively, a function $f$ is polynomial time approximable if, for most of $x$'s in the domain of $f$, $f(x)$ can be computed in polynomial time. So the function $g_{1/2}$ is polynomial time approximable since $M^x(n)$ converges in $O(n)$ steps for all $x$ such that $|x - 1/2| > 2^{-n}$. Similarly, $f(x) = 1/x$ on $(0,1]$ is polynomial time approximable since, for any $x > 2^{-n}$, $1/x$ has less than or
equal to \((n+1)\) digits in its integral part and its value may be obtained by long division in polynomial time. In the following we show that polynomial time computability is stronger than polynomial time approximability on continuous total functions.

**Theorem 3.1.** There exists a function \(f\) which is continuous on \([0,1]\), and is polynomial time approximable but is not in \(PF\).

**Proof.** Let

\[
f(x) = \begin{cases} 
  -1/(\log_2(x/4)) , & \text{if } 0 < x \leq 1 \\
  0 , & \text{if } x = 0.
\end{cases}
\]

Then \(f\) does not have a polynomially bounded modulus, because

\[
f(2^{-n}) = -1/(\log_2(x/4)) = 1/(n-2) , \text{ for all } n \in \mathbb{N}.
\]

So, \(f \notin PF\).

Now we show that \(f\) is polynomial time approximable.

We know that \(\log_2(1+x)\) is polynomial time computable on \([0,1]\) and assume that \(q\) is a polynomial bound of the time complexity of \(\log_2(1+x)\). Also, \(1/x\) is a polynomial time approximable function. Let \(p\) be a polynomial such that in \(p(n)\) steps, for all \(x \geq 2^{-n}\), \(1/x\) can be approximated to the \(n\)-th digit. Note that the computation of \(1/x\) and \(\log_2(1+x)\) can be easily extended to all \(x > 0\) using the fact that \(1/(2^kx) = (1/x)2^{-k}\) and \(\log_2(ab) = \log_2a + \log_2b\), and this modification takes only a linearly bounded amount of time.

Now we describe an algorithm for \(f\).
For given input \( n \), first ask the oracle to get \( d_1 \in \mathcal{D}_q(n+4) \) such that 
\[ |d_1 - x| < 2^{-q(n+4)}. \]

If \( d_1 < 2^{-n} \), then never halt.

Otherwise, compute \( d_2 \in \mathcal{D}_q(n+2) \) such that 
\[ |d_2 - \log_2(d_1/4)| < 2^{-q(n+2)}, \]
and then \( e \in \mathcal{D}_{n+2} \) such that 
\[ |e - 1/d_2| < 2^{-n(n+2)}. \]

Finally output \(-e\).

End of algorithm.

Since \( x < 1 \implies \log_2(x/4) \leq -2 \), we know that \( d_2 \leq -2 \) and
the computation of \( e \) by applying the approximation algorithm for
\( 1/x \) will halt in \( O(p(n+2)) \) steps, and so, the total time for
the above algorithm is \( O(q(n+2)) + O(p(n+2)) \).

Also, 
\[ |(-e) - f(x)| \]
\[ \leq |(-e) - (-1/d_2)| + |(-1/d_2) - (-1/(\log_2(d_1/4)))| \]
\[ + |(-1/\log_2(d_1/4)) - (-1/\log_2(x/4))| \]
\[ \leq 2^{-n+2} + \frac{1}{|d_2|} \cdot \frac{1}{|\log_2(d_1/4)|} \cdot |d_2 - \log_2(d_1/4)| \]
\[ + \frac{1}{|\log_2(d_1/4)|} \cdot \frac{1}{|\log_2(x/4)|} \cdot |\log_2(d_1/4) - \log_2(x/4)| \]
\[ \leq 2^{-n+2} + 2^{-n+2} + 2^{-n+2} < 2^{-n} \]

(since \( |\log_2(x/4)| \), \( |\log_2(d_1/4)| \) and \( |d_2| > 1 \))

We think that this is a realistic approach and may lead to
some interesting results about aplital functions. In particular, the
theory of recursive measure may be extended to the study of "polynomial time measurable sets".

In chapter 2 we studied the roots, the derivatives, the integrals, and the maximum values of polynomial time computable functions. We showed that a root of a polynomial time computable function is not necessarily polynomial time computable. Also, a polynomial time computable function may be nowhere differentiable. However, we were not able to answer the similar questions about the integrals and the maximum values: Is the integral (or, the maximum value) of a polynomial time computable function polynomial time computable? It is believed that there exist polynomial time computable functions $f$ and $g$ such that $\int_0^1 f \notin \text{PR}$ and $\max g \notin \text{PR}$. As shown in sections 2.5 and 2.6, however, a proof of the existence of such an $f$ or $g$ will show that $P \neq \text{PSPACE}$ or $P \neq \text{NP}$. Because $P \neq \text{PSPACE}$ and $P \neq \text{NP}$ are known to be hard problems, a reasonable concession is to correctly place the integral problem and the maximum value problem in the nondeterministic classes (e.g., $\text{NP}$-complete, $\text{NP}$-hard, $\text{PSPACE}$-complete) by relating them to known $\text{NP}$ problems. Unfortunately, it seems hard to relate known $\text{NP}$ combinatorial problems to these problems which have the "continuous" properties. An even weaker approach of relating such "continuous" problems to each other may be helpful in understanding their complexity. For example, we will know that the integral problem is at least as hard as the maximum value problem if we can find a polynomial time procedure which maps a function $f$ to a function $g$ with the property that $\int_0^1 g = \max f$. Such polynomial
time reducibility may help us to classify the hard problems about continuous functions and the study of these reducibilities may give us more insight into the complexity of "continuous" problems. Since the integration and the finding of maximum values are very basic problems, we feel that a careful study of the real-valued version of "P=NP?" is important and challenging.

In section 1.2, we used the reducibilities among left Dedekind cuts of real numbers to define the reducibilities among real numbers. From the point of view of real analysis, these reducibilities are not natural because we used the concept of computation of sets which is foreign to a real analyst. Instead, a natural approach is to treat real functions as operators on real numbers and then different topological properties of real functions may result in different types of operators.

Let $\mathcal{C}$ be a subclass of computable real functions.

**Definition 3.2.** Let $x$ and $y$ be two real numbers in $[0,1]$. We say that $x$ is $\mathcal{C}$-reducible to $y$ and write $x \leq_\mathcal{C} y$ if there is a real function $f \in \mathcal{C}$ such that $f(y) = x$.

For example, $\mathcal{C}$ may be any of the following classes.

$\mathcal{R} = \{\text{all partial recursive functions}\}$,

$\mathcal{T} = \{\text{all total recursive functions}\}$,

$\mathcal{PR} = \{\text{all primitive recursive functions}\}$,

$\mathcal{A} = \{\text{all recursive analytic functions}\}$,
\( \mathcal{B} = \{ \text{all recursive functions with bounded variations} \} \), or
\( \mathcal{M} = \{ \text{all monotone recursive functions} \} \).

By relating these "natural" reducibilities to the Turing, truth-table, and many-one reducibilities, we will be able to get a clearer picture of the structure of computable real numbers.

Many more problems are left untouched, for example, the recursive measure theory, the complexity of sequences of real numbers, and the complexity of differential equations, to name a few of them. We believe that the study of these problems will make contributions to numerical analysis and operations research as well as to complexity theory.
PART B

Polynomial Time Approximation to NP-hard Problems

Introduction

Many commonly encountered important problems appear to be intractable in the sense that no polynomial time algorithms are known for solving them. A large class of these intractable problems is shown to be "NP-hard" in the sense that the existence of a polynomial time algorithm for any problem in the class implies $P = NP$, which is widely believed to be false (cf. [27, 34, 38, 45]). Heuristic algorithms have been sought which were hoped to yield approximate solutions in polynomial time. Because of different goals and environments, different techniques have been used in the analysis of these heuristic algorithms. We can put the techniques into three categories.

The first is called the "guaranteed approximation" approach. Its goal is to find polynomial time algorithms which are guaranteed to give approximately optimal solutions to NP-hard optimization problems [36, 59]. This approach has been successfully applied to many problems such as the knapsack problem, the binpacking problem, and various job scheduling problems, and has yielded interesting results (cf. [36]). However, for many important problems, it has been shown that, unless $P = NP$, no polynomial time algorithms can solve the corresponding approximation problem [59]. These negative results seem to indicate
the restricted power of the guaranteed approximation approach.

The second approach, which is usually called the probabilistic approach, uses average case analysis to study the behavior of the approximation algorithms. It does not require an algorithm to always produce near-optimal solutions, but instead requires the algorithm to produce exact, optimal, or near-optimal solutions for "almost all" problem instances. In order to formulate the notion of "almost all", a probability distribution over the set of problem instances must be specified for each problem and then the average case analysis can be performed according to this specific distribution [46]. Many intuitively sound heuristic algorithms which do not guarantee approximately optimal solutions have been shown to have good performance under the average case analysis [46]. Another advantage of this approach over the guaranteed approximation approach is that the average case analysis can be applied to the problems which require definite "yes/no" answers whereas the worst case analysis can only be applied to optimization problems.

The main objection to this approach is that we seldom know which probability distribution is realistic (cf. [46]). Also, for different environments in which the same algorithm is used, different probability distributions may be required in order to get correct analyses of the algorithm. For many graphical problems, the theory of random graphs has been used to provide the required distribution of the input graphs [28, 46, 49]. The development of other realistic models of probability distribution needs further empirical studies.

The third approach is another type of a probabilistic approach
which uses probabilistic algorithms instead of deterministic algorithms. A probabilistic algorithm is a deterministic algorithm with the extra ability of "flipping-coins" during the computation in order to make decisions randomly. So, for a fixed problem instance, such an algorithm may produce different answers from different runs. The behavior of a probabilistic algorithm is naturally measured by the chance of producing correct or optimal solutions and this analysis does not depend on the distribution of the problem instances. The probabilistic algorithms have been shown to be more powerful than deterministic algorithms [39, 57]. In particular, Rabin [57] and Solovey and Strassen [62] have shown that the primality problem (which is known to be in \( \text{NP} \cap \text{co-NP} \) but not known to be in \( \text{P} \) [55, 56]) can be solved in polynomial time by a probabilistic algorithm with arbitrarily small chance of error. However, not many intractable problems are known to be solvable by probabilistic algorithms in polynomial time.

In this part of the paper, we will study some important questions about approximations to intractable problems. In chapter 4 we first use recursive functions as abstract models of optimization problems and show that, under this model, unless \( \text{P} = \text{NP} \), some optimization problems are not approximable at all. Then, an approximation preserving reducibility is proposed to classify the \( \text{NP} \)-hard combinatorial problems according to their approximability (in the sense of worse case approximation). We demonstrate the practicality of the reducibility by applying it to some important problems. In particular, some natural reductive
functions between NP-hard problems which preserve both the number of solutions and the optimal solutions do not preserve approximate solutions. Thus, unlike Simon's parsimonious reducibility [61] and Lynch and Lipton's structure preserving reducibility [52], this reducibility can distinguish the problems with similar nature to some extent. We also point out some practically important problems as the candidates for complete problems in the nonapproximable class or the partially approximable classes.

In chapter 5 we study probabilistic algorithms for NP-hard problems by using probabilistic Turing machines as a model. The basic properties of probabilistic Turing machines have been studied by Gill [39]. We use his result to show a strong negative result that for many NP-hard combinatorial problems including the traveling salesman problem, the graph coloring problem and the maximum clique problem, it is not likely that we can find any polynomial time probabilistic algorithms for them. Therefore, despite the fact that some intractable problem such as the primality problem (which is not known to be NP-complete) can be solved by a probabilistic algorithm in polynomial time, we should not expect too much from the probabilistic approach. The relationship between the class of polynomial time probabilistic solvable problems and the polynomial hierarchy of Meyer and Stockmeyer [63] is also studied.

In chapter 6 the average case analysis of polynomial time approximation is studied from an abstract point of view. A definition of the density of sets which was first suggested by Meyer [53] and Lynch
[50] is adopted to formalize the notion of a "95% correct" approximation. We first consider the concept of creativity and simplicity at the polynomial time level and define p-creative sets and p-simple sets. It is shown that there is no exponential time computable p-creative set and there exists a $\leq_{tt}^p$-complete p-simple set. With these results we are able to show that there is no relationship between the $\leq_{tt}^p$-completeness of an exponential time computable set and its approximability. We do know that such a set must have an easy part which is at least infinite, but whose complement is larger than "p-sparse" sets. In conclusion, we discuss some open questions. In particular, the average case analysis of approximation algorithms for NP-complete sets is discussed.

In part B, let $\Sigma$ be a finite set of symbols. We write $\Sigma^*$ to denote all finite strings of symbols over $\Sigma$. A language is a subset of $\Sigma^*$. For the nondeterministic Turing machines and their time complexity, we will follow the definitions in [27]. For the oracle Turing machines, we will use the definitions given in part A. The classes of languages of particular interest are defined as following:

Let $T$ be a function and $\mathcal{C}$ a class of functions.

$$
\text{DTIME}(T) = \left\{ L \subseteq \Sigma^* \mid \text{there is a deterministic Turing machine (DTM) } M \text{ of time complexity } \leq T \text{ accepting } L \right\},
$$

$$
\text{NTIME}(T) = \left\{ L \subseteq \Sigma^* \mid \text{there is a nondeterministic Turing machine (NTM) } M \text{ of time complexity } \leq T \text{ accepting } L \right\},
$$
$\text{DTIME}(\mathcal{C}) = \bigcup_{T \in \mathcal{C}} \text{DTIME}(T)$,
$\text{NTIME}(\mathcal{C}) = \bigcup_{T \in \mathcal{C}} \text{NTIME}(T)$,

$P = \text{DTIME}(\text{Polynomials})$,
$NP = \text{NTIME}(\text{Polynomials})$,
$\text{co-NP} = \{ L \subseteq \Sigma^* \mid \overline{L} \in NP \}$,
$\text{EXPTIME} = \text{DTIME}(2^{\text{linear}})$.

The polynomial time reducibilities are those in $[47]$. Namely,

$\leq^p_T$: polynomial time Turing reducibility,
$\leq^p_{tt}$: polynomial time truth-table reducibility,
$\leq^p_m$: polynomial time many-one reducibility.

We say that a set $L \subseteq \Sigma^*$ is NP-hard if, for all $L' \in \text{NP}$, $L' \leq^p_m L$, and NP-complete if $L$ is NP-hard and $L \in \text{NP}$. If $L_1 \leq^p_T L_2$, then we write $L_1 \in P(L_2)$ or $L_1 \leq^p L_2$. If $L_1 \leq^p_T L_2$ and $L_2 \in \mathcal{C}$, then we may write $L \in P(\mathcal{C})$.

The definitions of optimization problems discussed in this part can be found in the Appendix.
Chapter IV

Approximation Preserving Reducibility

In this chapter we will concentrate on optimization problems and the worst case analysis of approximation algorithms. The class of optimization problems which we are mostly interested in includes the travelling salesman problem, the graph coloring problem, the maximum clique problem, the binpacking problem, and the knapsack problem, to name a few. These problems have been characterized as NP-hard [27, 34, 36, 45, 48] and the corresponding approximation problems have been well-studied [36, 59]. As pointed out by Garey and Johnson [36], the NP-hard optimization problems may be classified into three subclasses according to the nature of the best worst case performance: namely, the subclass of problems for which there exist polynomial time approximation schemes capable of guaranteeing performance arbitrarily close to optimal solutions, the subclass of problems for which there exists a best bound of approximation so that some polynomial time algorithm satisfies the bound and no such algorithm satisfies a better bound, and the subclass of problems for which no polynomial time algorithm satisfies any bound. This classification "reflects the limitations of our current knowledge as much as any inherent differences between the problems" [36]. Under polynomial time many-one reducibilities, all NP-complete problems are considered
to be equivalent. What other tools can reflect the inherent differences between these $\text{NP}$-complete problems and help us to distinguish them? Some strong reducibilities have been suggested to answer this question [29, 52, 61]. However, the reducibilities proposed although providing more insight into the problems do not separate the problems into classes determined by the best performance bounds. In this chapter, we define a very strong reducibility called approximation preserving reducibility which preserves the above mentioned classification and study the properties of this reducibility.

4.1. Recursive Functions as Models of Optimization Problems

We first study the problems at the "formal language level" by using recursive functions as models of problems and define the approximation preserving reducibility on functions.

A simple way of stating an optimization problem is to treat it as a language recognition problem which, for a given problem instance and a number, requires a "yes/no" answer according to whether the given number is the measure of the optimal solutions of the given problem instance. For example, the graph coloring problem is encoded as a set of order pairs each of which consists of a graph and a number which might be the smallest number of colors needed to color the graph. This is not useful in practice. We may not know how large the minimum number of colors is and may want to know the actual coloring in addition to the number of colors used. So, we consider the problems as functions from $\Sigma^*$ to $\Sigma^*$.
Notation. [55] All classes of functions are *'ed. For example, NP* is the class of functions which are computable in polynomial time by nondeterministic Turing machines.

Definition 4.1. A function \( f \) is in \( P^*(NP^*) \) if there exists a deterministic (nondeterministic) Turing machine \( M \) which computes \( f \) with time complexity bounded by a polynomial.

A language \( L \) is said to be proper-\( \Delta_2^P \) if \( L \in \Delta_2^P = P(NP) \) def and \( L \notin NP \cup co-NP \) unless \( NP = co-NP \). When the optimization problems are considered to be language recognition problems, most of them have been shown to be proper-\( \Delta_2^P \) [48]. So, we need to consider functions of higher complexity.

Definition 4.2. Let \( f \) be a recursive function.

(i) \( f \in \Delta_2^{P,*} \) or, \( f \in P^*(NP) \), if and only if there exist an oracle Turing machine (OTM) \( M \) and a set \( A \in NP \) such that \( M^A \) computes \( f \) with polynomial time complexity.

(ii) \( f \in \Sigma_2^{P,*} \) or, \( f \in NP^*(NP) \), if and only if there exist an oracle nondeterministic Turing machine (QNTM) \( M \) and a set \( A \in NP \) such that \( M^A \) computes \( f \) with polynomial time complexity.

Meyer and Stockmeyer [63] have constructed a polynomial time hierarchy as an analogue of Kleene's arithmetical hierarchy. A similar hierarchy of functions may be easily defined by the notion of relative computation. However, we will not study this here because the most interesting problems are in \( P^*(NP) \) and \( NP^*(NP) \).
In the following, we study the relationship between functions in \( P^*(NP) \) or \( NP^*(NP) \) and sets in \( P \) or \( NP \).

By \( g(x,y) \downarrow \), we mean that \( g(x,y) \) converges. Otherwise, we write \( g(x,y) \uparrow \) to denote that, in polynomial time, \( g(x,y) \) outputs a special symbol in \( \Sigma \), say, \( \# \), instead of strings of symbols in \( \Sigma^* \). We write \( \lg(x) \) or \(|x|\) to express the length of the string \( x \in \Sigma^* \).

**Definition 4.3.** A function \( f \) is polynomial length bounded if there is a polynomial \( p \) such that

\[
( \forall x ) \ [ \lg(f(x)) \leq p(\lg(x))] .
\]

**Definition 4.4.** Let \( f \) be a function. Then,

\[
G_f = \{(x,y) \mid f(x) = y\} \quad \text{(graph of } f \text{)}, \quad \text{and}
\]

\[
P_f = \{(x,y) \mid f(x) \leq y\} \quad \text{(projection of } f \text{)}.
\]

**Theorem 4.1.** (i) \( A \in \Delta^P_2 \) if and only if \( \chi_A \in \Delta^{P,*}_2 \).

(ii) \( f \in \Delta^{P,*}_2 \) if and only if \( f \) is polynomial length bounded and \( P_f \in \Delta^P_2 \).

**Proof.** (i) is evident.

(ii). The computation of \"\((x,y) \in P_f ?\)" can be done by first computing \( f(x) \) then checking whether \( f(x) \leq y \). The computation of \( f(x) \) can be done by performing the binary search over \( y \) in \( \{0, 1, 2, \ldots, 2^p(\lvert x \rvert)\} \) such that \((x,y) \in P_f \) and \((x,y-1) \notin P_f \), where \( p \) is a polynomial served as a length bound for \( f \).
Theorem 4.2. [55] The following are equivalent.

(i) \( f \in \text{NP}^* \);
(ii) \( f \) is polynomial length bounded and \( G_f \in \text{NP} \);
(iii) \( f \) is polynomial length bounded and \( G_f \in \text{NP} \cap \text{co-NP} \);
(iv) \( f \) is polynomial length bounded and \( P_f \in \text{NP} \cap \text{co-NP} \);
(v) There exist a function \( g \in \text{P}^* \) and a polynomial \( p \) such that for any \( x \),

\[
(a) \quad (\forall y)_{p(|x|)} \left[ g(x,y) \downarrow \Rightarrow g(x,y) = f(x) \right], \quad \text{and}
\]

\[
(b) \quad (\exists y)_{p(|x|)} \left[ g(x,y) \uparrow \right],
\]

where \( (\exists y)_n R(y) \) means that there exists a string \( y \) of length \( n \) which satisfies the property \( R \). The meaning of \( (\forall y)_n R(y) \) is similar.

Proof. The equivalence of (i) \( \Rightarrow \) (iv) is in [55] and is simple. We sketch the proofs of (ii) \( \Rightarrow \) (v) and (v) \( \Rightarrow \) (ii):

(ii) \( \Rightarrow \) (v). Assume that \( G_f \in \text{NP} \) and \( f \) is polynomial length bounded. Then, there is a polynomial time predicate \( R \) and a polynomial \( q \) such that

\[
(x,y) \in G_f \Rightarrow (\exists z)_{q(|x|)} R(x,y,z).
\]

Let

\[
g(x,y,z) = \begin{cases} 
    y, & \text{if } R(x,y,z), \\
    \uparrow, & \text{otherwise}.
\end{cases}
\]

and \( p = q + p_f \). Then \( g \in \text{P}^* \).

If \( g(x,y,z) \downarrow \) and \( |y,z| \leq p(|x|) \), then \( R(x,y,z) \). But this means that \( (x,y) \in G_f \), or \( y = f(x) \). So, (a) is satisfied. Now, since \( f \) is total, for any \( x \), there is a \( y \), \( |y| \leq p_f(|x|) \), such
that \((x,y) \in G_f\), and (b) follows.

\((v) \Rightarrow (ii).\) Assume that \(g\) and \(p\) satisfy the conditions (a) and (b). Since \(g \in P^*\), it follows that \(f\) is polynomial length bounded.

To see that \(G_f \in NP\), we consider the following algorithm:

For given \((x,y)\), guess \(z\), \(|z| \leq p(|x|)\), and compute \(g(x,z)\). If \(g(x,z)\downarrow\) and output \(y\), then answer "yes"; otherwise, answer "no". \(\square\)

**Corollary 4.2.1.** The following are equivalent.

(i) \(f \in NP^*(NP)\);

(ii) \(f\) is polynomial length bounded and \(G_f \in NP(NP)\);

(iii) \(f\) is polynomial length bounded and \(G_f \in NP(NP) \cap \text{co-NP}(NP)\);

(iv) \(f\) is polynomial length bounded and \(P_f \in NP(NP) \cap \text{co-NP}(NP)\);

(v) there exist a function \(g \in P^*\) and a polynomial \(q\) such that, for any \(x\),

\[
(a) \quad (\forall y)_{p(|x|)} \left( \forall z \right)_{p(|x|)} \left[ g(x, y, z)\downarrow \Rightarrow g(x, y, z) = f(x) \right]
\]

\[
(b) \quad (\exists y)_{p(|x|)} \left( \forall z \right)_{p(|x|)} \left[ g(x, y, z) \downarrow \right].
\]

**Proof.** The proof is similar to that of the theorem and is omitted here. \(\square\)

From theorems 4.1 and 4.2 and the fact that the language recognition form of the graph coloring problem is a proper-\(\Delta^p_2\) problem we know that its function form is proper-\(\Delta_{2}^{p,*}\) (in \(\Delta_{2}^{p,*} \subseteq NP^*\)). So are other optimization problems proper-\(\Delta^p_2\). So, the class \(\Delta_{2}^{p,*}\) is the most interesting class for us.
A polynomial time approximation algorithm for function $f \in \Delta^P_2$ is just a function $g \in P^*$ whose function values are close to that of $f$.

**Definition 4.5.** (i) A function $g$ $\varepsilon$-approximates function $f$ if

$$(\forall x) \left[ |f(x) - g(x)| \leq \varepsilon \cdot f(x) + d \right].$$

(ii) A function $g$ $\varepsilon$-approximates $f$ if there exists a number $d < \max_{x \in \mathbb{N}} f(x)$ such that $g$ $\varepsilon$-approximates $f$.

(iii) We say that $f$ is polynomial time $\varepsilon$-approximable, or, $f \in \varepsilon$-AP, if there is a function $g \in P^*$ such that $g$ $\varepsilon$-approximates $f$.

(iv) We say that $f$ is polynomial time arbitrarily approximable, or $f \in P^*$, if $f \in \varepsilon$-AP for all $\varepsilon > 0$.

(v) We say that $f$ is fully polynomial time approximable, or $f \in \text{FAP}$, if there is a function $g \in P^*$ such that, for all $n \in \mathbb{N}$, the function $\lambda x [g(n,x)] \ (1/n)$-approximates $f$.

Now, we formally show that the above mentioned subclasses of problems are not empty by actually constructing such functions. Furthermore, we show that any $\varepsilon$, $0 < \varepsilon < 1$, may serve as a best bound of approximation for some function $f$ in $\Delta^P_2$.

**Theorem 4.3.** (i) There exists a proper-$\Delta^P_2$ function $f$ such that for any $\varepsilon > 0$, there is a function $g \in P^*$ such that $g$ $\varepsilon$-approximates $f$.

(ii) For any $\varepsilon_0$, $0 < \varepsilon_0 < 1$, there exists a proper-$\Delta^P_2$
function $f$ such that

(a) there is a function $g \in P^*$ which $\varepsilon_0$-approximates $f$,

(b) for any $\varepsilon$, $0 < \varepsilon < \varepsilon_0$, and any $g' \in P^*$, $g'$ does not $\varepsilon$-approximate $f$, unless $P = NP$.

(iii) There exists a proper-$\Delta^P_2$* function $f$ such that for any $\varepsilon$, $0 < \varepsilon < 1$, and any function $g \in P^*$, $g$ does not $\varepsilon$-approximate $f$, unless $P = NP$.

Proof. (i) The knapsack problem is a good example [42]. Let $A$ be an NP-complete set. Define

$$f(x) = \begin{cases} x, & \text{if } x \in A. \\ x+1, & \text{if } x \notin A. \end{cases}$$

Then $f$ is proper-$\Delta^P_2$* ($f \in NP^* \Rightarrow [A \leq^P_m G_f \land \overline{A} \leq^P_m G_f \land G_f \in NP \land co-NP] \Rightarrow NP = co-NP$). And, the identity function $e$, $e(x) = x$, $\varepsilon$-approximates $f$ for all $\varepsilon$.

For any $\varepsilon > 0$, take $x_0 = \lceil 1/\varepsilon \rceil$, then the function

$$g(x) = \begin{cases} x, & \text{if } x \geq x_0 \text{ or } [x < x_0 \land x \in A]. \\ x+1, & \text{if } x < x_0 \land x \notin A. \end{cases}$$

is in $P^*$ and $|g(x) - f(x)| = 0$, if $x < x_0$; and $|g(x) - f(x)| = 1 < \varepsilon \cdot x \leq \varepsilon \cdot f(x)$, if $x \geq x_0$. I.e., $g$ $\varepsilon$-approximates $f$.

(ii) The binpacking problem is an example in this class [36,44]. We call this class PAP. We assume that functions can take rational values. Let $A$ be an NP-complete set. For given $\varepsilon_0 \in Q^+$, let $r = (1+\varepsilon_0)/(1-\varepsilon_0)$ and
\[ f(x) = \begin{cases} x, & \text{if } x \in A, \\ r \cdot x, & \text{if } x \notin A. \end{cases} \]

Similar to the function in (i), \( f \) is proper-\( \Delta_2^P \). Let

\[ g(x) = (1 + \varepsilon_0) \cdot x. \]

Then, for \( x \in A \), \( |f(x) - g(x)| = \varepsilon_0 \cdot x = \varepsilon_0 \cdot f(x) \), and for \( x \notin A \), \( |f(x) - g(x)| \leq (r - (1 + \varepsilon_0)) \cdot x \leq \varepsilon_0 \cdot f(x) \).

So, \( g \) \( \varepsilon_0 \)-approximates \( f \).

Now, by way of contradiction, assume that \( g' \) \( \varepsilon \)-approximates \( f \) for some \( \varepsilon < \varepsilon_0 \). Then, for all \( x \), we have

\[ |g'(x) - f(x)| \leq \varepsilon \cdot f(x) + d. \]

So,

\[ x \in A \Rightarrow g'(x) \leq f(x) + \varepsilon \cdot x + d = (1 + \varepsilon) \cdot x + d. \]

Similarly,

\[ x \notin A \Rightarrow g'(x) \geq f(x) - \varepsilon \cdot f(x) - d = (1 - \varepsilon) \cdot (1 + \varepsilon_0)/(1 - \varepsilon_0) \cdot x - d. \]

For \( x > 2d/(\varepsilon_0 - \varepsilon) \), we have

\[ (1 + \varepsilon) \cdot x + d < (1 + \varepsilon_0) \cdot x - d \leq (1 - \varepsilon) \cdot r \cdot x - d. \]

So, for \( x \) large enough,

\[ x \in A \iff g'(x) \leq (1 + \varepsilon) \cdot x + d. \]

Hence, \( A \) is in \( P \), or, \( P = NP \).

(iii) The travelling salesman problem is an example \([59]\). We call this class \( NAP \). Let \( A \) be an \( NP \)-complete set, and

\[ f(x) = \begin{cases} x, & \text{if } x \in A, \\ 0, & \text{if } x \notin A. \end{cases} \]
Then, obviously, $f$ is proper-$\Delta^p_2$.

Assume, by way of contradiction, that $g \in \mathcal{P}^*$ $\varepsilon$-d-approximates $f$ for some $0 < \varepsilon < l$. Then,

$$x \in A \Rightarrow |g(x) - f(x)| \leq \varepsilon \cdot f(x) + d$$
$$\Rightarrow g(x) \geq x - \varepsilon \cdot x - d = (1 - \varepsilon) x - d,$$ and

$$x \notin A \Rightarrow |g(x) - f(x)| \leq \varepsilon \cdot f(x) + d$$
$$\Rightarrow g(x) \leq d.$$ 

So, we have $x \in A \iff g(x) > d$, for all $x > 2d/(1-\varepsilon)$.

Hence $A \in \mathcal{P}$, or, $P = \mathcal{NP}$.$\Box$

As pointed out by Sahni and Gonzalez [59], the functions constructed in the above proof are "unnatural" and are not very interesting. However, it should be interesting from the theoretical standpoint that any situation may happen when worst case analysis of approximation algorithms is performed. Also, this shows the soundness of our adoption of recursive functions as models.

4.2. Definition

Now we consider reducibilities among functions.

As defined in part A, we write $M^f$ to denote the function computed by $M$ with function-oracle $f$.

Definition 4.6. (i) We say that $f$ is Turing reducible to $g$ and write $f \leq_T g$ if there is an oracle Turing machine (OTM) $M$ such that $M^g$ computes $f$. 
(ii) We say that \( f \) is many-one reducible to \( g \) and write \( f \leq_m g \) if there exist two recursive functions \( \varphi \) and \( \psi \) such that 
\[
(\forall x) \ f(x) = \psi(g(\varphi(x)))
\]
We write \( f \leq_m g \) via \( (\varphi, \psi) \) to emphasize that \( \varphi \) and \( \psi \) are reductive functions.

**Definition 4.7.** (i) \([55]\) \( f \) is polynomial time Turing reducible to \( g \), or \( f \leq_T^p g \), if \( f \leq_T g \) and the reductive machine has polynomial time complexity.

(ii) \( f \) is polynomial time many-one reducible to \( g \), or \( f \leq_m^p g \), if \( f \leq_m g \) via \( (\varphi, \psi) \) and both \( \varphi \) and \( \psi \) are polynomial time computable.

It is well-known that \( f \) being Turing reducible to \( g \) is equivalent to \( G_f \) being enumeration reducible to \( G_g \). This fact has been extended to the polynomial level by Selman \([60]\). He defined a relation between sets called polynomial time enumeration reducibility and denote it by \( \leq_{pe} \). Then, \( f \leq_T^p g \) if and only if \( G_f \leq_{pe} G_g \).

It is easy to see that \( f \leq_m^p g \) implies \( f \leq_T^p g \). Now we define a much stronger reducibility than \( \leq_m^p \).

**Definition 4.8.** A function \( \varphi \) is said to be relatively continuous if there is a function \( m : [0,1] \rightarrow \mathbb{R}^+ \) which is continuous, increasing, and \( m(0) = 0 \), such that
\[
(\forall \varepsilon, 0 < \varepsilon < 1) (\forall d) (\exists d') \quad |x - y| \leq \varepsilon \cdot x + d \Rightarrow |\varphi(x) - \varphi(y)| \leq m(\varepsilon) \cdot \varphi(x) + d'.
\]
**Definition 4.2.** \( f \) is polynomial time approximation reducible to \( g \), or \( f \preceq^P_a g \), if there exist two functions \( \varphi \) and \( \psi \) such that \( f \preceq_m g \) via \((\varphi, \psi)\) and \( \psi \) is relatively continuous.

The meaning of the polynomial time approximation reducibility can be seen clearer from the following diagram.

\[
\begin{array}{c}
x \xrightarrow{\varphi} \varphi(x) \xrightarrow{g} g(\varphi(x)) \approx y \\
\end{array}
\]

The next theorem follows immediately from the above diagram.

**Theorem 4.4.** If \( f \preceq^P_a g \), then

(i) \( g \in \varepsilon\text{-AP} \Rightarrow f \in \varepsilon'\text{-AP} \) for some \( \varepsilon' > 0 \).

(ii) \( g \in \text{AP} \Rightarrow f \in \text{AP} \).

Note that the membership in FAP is not necessarily preserved by \( \preceq^P_a \) because the "modulus of continuity" of \( \psi \) is not known. In practice, the modulus functions \( m \) of many \( \psi \)'s are just the identity function and then the membership in FAP will be preserved.

**Theorem 4.5.** \( \preceq^P_a \) is reflexive and transitive.
Proof. Reflexivity of \( \lesssim^p_a \) is clear.

To see the transitivity, we assume that \( f \lesssim^p_a g \) via \((\varphi_1, \psi_1)\)
and \( g \lesssim^p_a h \) via \((\varphi_2, \psi_2)\). Then \( f \lesssim^p_m h \) via \((\varphi_2 \circ \varphi_1, \psi_2 \circ \psi_1)\).
Assume that \( m_1 \) and \( m_2 \) are the continuous functions on \([0,1]\)
which give the error bounds for \( \psi_1 \) and \( \psi_2 \), respectively. Then,

\[
|x - y| \leq \varepsilon \cdot x + d
\]

\[
\Rightarrow |\psi_2(x) - \psi_2(y)| \leq m_2(\varepsilon) \cdot \psi_2(x) + d', \quad \text{for some } d',
\]

\[
\Rightarrow |\psi_1 \circ \psi_2(x) - \psi_1 \circ \psi_2(y)| \leq m_1 \circ m_2(\varepsilon) \cdot \psi_1 \circ \psi_2(x) + d'', \quad \text{for some } d''. \]

4.3. Application

In this section we want to apply the reducibility \( \lesssim^p_a \) to
natural problems and show how the classification of the problems
according to the best bound of performance can be done by using \( \lesssim^p_a \).

We will discuss the problem at "the abstract problem level"
by using Johnson's framework for problem settings \([43]\), which is
stated in the following.

An optimization problem \( S \) consists of

1. a set \( \text{INPUT}_S \) of possible inputs,
2. a map \( \text{SOL}_S \) which maps each \( u \in \text{INPUT}_S \) to a finite set of
   approximate solutions, and
3. a measure \( m_S : \text{SOL}_S(\text{INPUT}_S) \rightarrow \mathbb{Q}^+ \) defined for all possible
   approximate solutions.

The optimal measure \( u_S^* \) then is defined by
\[ u^*_S = \text{optimal } \{ m_S(x) \mid x \in \text{SOL}_S(u) \} \]

and the optimal solution is

\[ \text{CPSOL}_S(u) = \{ x \in \text{SOL}_S(u) \mid u^*_S = m_S(x) \} . \]

The corresponding \( \preceq^P_a \) at the abstract problem level can also be stated in this form.

**Definition 4.10.** Assume that \( R \) and \( S \) are two optimization problems. We say that \((\varphi, \psi)\) polynomially reduces \( R \) to \( S \) and preserves approximate solutions if

(i) \( \varphi : \text{INPUT}_R \rightarrow \text{INPUT}_S \) is polynomial time computable.

(ii) (preserving solutions) \( \psi : \text{SOL}_S(\varphi(\text{INPUT}_R)) \rightarrow \text{SOL}_R(\text{INPUT}_R) \)

is polynomial time computable and

\[ (\forall u \in \text{INPUT}_R)[ \psi(\text{SOL}_S(\varphi(u))) = \text{SOL}_R(u) ] . \]

(iii) (preserving optimal solutions)

\[ (\forall u \in \text{INPUT}_R) \ (\forall x \in \text{SOL}_S(\varphi(u))) \]

\[ [ m_S(x) = (\varphi(u))^*_S \Rightarrow m_R(\psi(x)) = u^*_R ] . \]

(iv) (preserving approximate solutions) there exists a continuous and increasing function \( m : [0,1] \rightarrow [0,1] \) such that \( m(0) = 0 \), \( m(1) = 1 \), and \( (\forall \varepsilon, 0 < \varepsilon < 1) \ (\forall d \geq 0) \ (\forall u \in \text{INPUT}_R) \)

\[ (\forall x \in \text{SOL}_S(\varphi(u))) \ (\exists d' \geq 0) \]

\[ \left[ |m_S(x) - (\varphi(u))^*_S| \leq \varepsilon \cdot (\varphi(u))^*_S + d \right. \]

\[ \left. \Rightarrow |m_R(\psi(x)) - u^*_R| \leq m(\varepsilon) \cdot u^*_R + d' \right] . \]

Using this abstract problem form, we apply the reducibility \( \preceq^P_a \) to some natural problems found in [27] and [45].
Theorem 4.6.

(i) Max clique $\equiv^p_a$ Set packing.
(ii) Min vertex cover $\leq^p_a$ Feedback vertex.
(iii) Min vertex cover $\leq^p_a$ Set covering.
(iv) Max satisfiability $\leq^p_a$ Max clique.
(v) Graph coloring $\equiv^p_a$ Min clique cover.
(vi) 2-partition $\leq^p_a$ Sum of subsets.

Proof. For all the problems in (i) - (vi), the "natural" reductions used in [45] or [27] preserve the approximate solutions (see also [29]). We check (iii) here.

Let $VC$ denote the minimum vertex cover problem, and $SC$ denote the set cover problem. Then,

\begin{align*}
INPUT_{VC} &= \{ G = (V, E) \mid G \text{ is a simple, finite graph with the set of vertices } V \text{ and the set of edges } E \}, \\
SOL_{VC}(G) &= \{ V' \subseteq V \mid (\forall e \in E)(\exists v \in V') v \text{ is incident with } e \}, \\
m_{VC}(V') &= |V'| = \text{the number of the vertices in } V'.
\end{align*}

and

\begin{align*}
INPUT_{SC} &= \{ F = \{ S_1, \ldots, S_p \} \mid F \text{ is a finite collection of finite sets} \}, \\
SOL_{SC}(F) &= \{ F' \subseteq F \mid U \{ S \mid S \in F' \} = U \{ S \mid S \in F \}, \\
m_{SC}(F') &= |F'|
\end{align*}

The natural reductive functions $\varphi$ and $\psi$ are:

\begin{align*}
\varphi(G) &= \varphi(G = (V, E) = (\{ v_1, \ldots, v_p \}, E)) = \{ S_1, \ldots, S_p \} \\
&= \{ e \in E \mid v_i \text{ is incident with } e \}, i = 1, \ldots, p.
\end{align*}

\begin{align*}
\psi(F') &= \psi(F' = \{ S_{k_1}, \ldots, S_{k_m} \}) = \{ v_{k_1}, \ldots, v_{k_m} \}.
\end{align*}
We must check the conditions (i) - (iv) in definition 4.10 in order to show that \((\varphi, \psi)\) does preserve the required properties.

(i) The polynomial time computability of \(\varphi\) and \(\psi\) is clear.

(ii) Let \(G = (V,E) = (\{v_1, \ldots, v_p\}, E) \in \text{INPUT}_{VC}\). Then, 
\[\varphi(G) = \{S_1, \ldots, S_p\}, \text{ where } S_i = \{e \in E | v_i \text{ is incident with } e\},\]
\(i = 1, \ldots, p\). Assume that \(\{S_{k_1}, \ldots, S_{k_m}\} \subseteq \{S_1, \ldots, S_p\}\) is a solution of \(\varphi(G) = \{S_1, \ldots, S_p\}\). I.e., \(\bigcup_{i=1}^{m} S_{k_i} = \bigcup_{i=1}^{p} S_i = E\).
Then, for any \(e \in E\), there is an \(i\) such that \(e \in S_{k_i}\). Or, there is an \(i\) such that \(v_{k_i}\) is incident with \(e\). So, \(\psi(\{S_{k_1}, \ldots, S_{k_m}\}) = \{v_{k_1}, \ldots, v_{k_m}\} \in \text{SOL}_{VC}(G)\).

Conversely, assume that \(\{v_{k_1}, \ldots, v_{k_m}\} \subseteq V\) is in \(\text{SOL}_{VC}(G)\). Then, the corresponding set \(\{S_{k_1}, \ldots, S_{k_m}\} \subseteq \{S_1, \ldots, S_p\}\) is in \(\text{SOL}_{SC}(\varphi(G))\) by the same argument. Now, \(\psi(\{S_{k_1}, \ldots, S_{k_m}\}) = \{v_{k_1}, \ldots, v_{k_m}\}\). Thus, we have shown
\[\psi(\text{SOL}_{SC}(\varphi(G))) = \text{SOL}_{VC}(G)\].

(iii) & (iv). Since the mapping \(\psi\) preserves the size of the solutions, i.e., \(|\psi(F')| = |F'|\), we know that the optimal solution is preserved. The approximate solutions are also preserved:
\[|m_{SC}(F') - (F = \varphi(G))^*_{SC}| \leq \varepsilon \cdot F^*_{SC} + d\]
\[\Rightarrow |m_{VC}(\psi(F')) - G^*_{VC}| \leq \varepsilon \cdot G^*_{VC} + d\]
since \(m_{SC}(F') = m_{VC}(\psi(F'))\) and \(G^*_{VC} = (\psi(G))^*_{SC}\). \(\square\)
Note that we did not include

\[
\text{Max clique } \leq^D_m \text{ Min vertex cover}
\]
in our list. It is easily seen that by constructing complementary graphs we can transform one problem to the other and preserve solutions and optimal solutions. Although this reduction also preserves approximate solutions, it does not preserve the worst case bounds of error. For instance, let \( G \) be a graph of size 100 and its largest clique be of size 20. Then, given an algorithm for the min vertex cover problem with guaranteed error bound \( 1/8 \), we can transform it into an algorithm for the max clique problem which finds a clique of \( G \) of size 10 and has a bigger error bound \( 1/2 \) on \( G \). For the graphs of larger size with smaller cliques, the error bound may be worse. So, as far as the worst case error bound is concerned, the natural reductive functions do not satisfy the condition (iv) of definition 4.10. Since one of these two problems is a maximization problem and the other is a minimization problem, we expect such a result to reflect their difference.

Simon [61] considered parsimonious reductions which preserve "the number of solutions". He showed that most interesting problems mentioned in [27, 45] are parsimoniously related. With a few exceptions the reductions already specified are parsimonious reductions. Lynch and Lipton [52] considered reductions on relations rather than sets and defined reducibility \( \leq^D \) which is similar to our \( \leq^D_m \) and is stronger than parsimonious reduction. However, \( \leq^D \) is still not strong enough to distinguish the problems with some
different nature. For instance, the exact cover problem is \( \leq^p \)
reducible to the sum of subsets problem which is known to be in AP.
However, the approximation algorithm for the sum of subsets problem
does not seem modifiable into an approximation algorithm for the exact
cover problem. In the following we show that the natural reduction
which \( \leq^p \)-reduces the exact cover problem to the sum of subsets problem
does not preserve approximate solutions.

Let \( \text{EC} \) denote the exact cover problem and \( \text{SS} \) denote the
sum of subset problem. Then,

\[
\text{INPUT}_{\text{EC}} = \{ F = \{ S_1, \ldots, S_p \} \mid F \text{ is a finite collection of finite sets} \},
\]

\[
\text{SOL}_{\text{EC}}(F) = \{ F' \subseteq F \mid U \{ S \mid S \in F \} = U \{ S \mid S \in F' \} \},
\]

\[
m_{\text{EC}}(F') = \Sigma_{S \in F'} |S|
\]

and

\[
\text{INPUT}_{\text{SS}} = \{ <T, s, b> \mid T \text{ is a finite set, } s : T \rightarrow \mathbb{Q}^+, \ b \in \mathbb{Q}^+ \}.
\]

\[
\text{SOL}_{\text{SS}}(<T, s, b>) = \{ T' \subseteq T \mid \Sigma_{t \in T'} s(t) \geq b \},
\]

\[
m_{\text{SS}}(T') = \Sigma_{t \in T'} s(t).
\]

Consider the following reduction suggested by [45].
If \( F = \{ S_1, \ldots, S_p \} \), and \( \bigcup_{i=1}^{p} S_i = \{ u_1, \ldots, u_t \} \) then

\[
\varphi(F) = <T, s, b>, \text{ where}
\]

\[
T = \{ t_1, \ldots, t_p \},
\]

\[
s(t_j) = \Sigma_{u_i \in S_j} (1+p)^{i-1}, \quad j = 1, \ldots, p, \text{ and}
\]

\[
b = ( (1+p)^t - 1 ) / p,
\]
and $\psi\left( \{ t_{k_1}, \ldots, t_{k_m} \} \subseteq T \right) = F^* = \{ S_{k_1}, \ldots, S_{k_m} \}$.

It is not hard to check that the cleverly assigned values $s(t_j)$'s separate the elements $u_i$'s so that

$$\bigcup_{i=1}^m S_{k_i} = \{ u_1, \ldots, u_t \} \iff \sum_{i=1}^m s(t_{k_i}) \geq b.$$ 

That is, we have shown that $(\varphi, \psi)$ preserves solutions. However, it is clear that the optimal solutions and approximate solutions are not preserved because the elements $u_1, \ldots, u_t$ are of the same weight but the corresponding elements are of the different weights $(1+p)^0, (1+p)^1, \ldots, (1+p)^t-1$. For example, let both

$$\{\{u_1, u_2\}, \{u_1, u_3, u_4, \ldots, u_t\}\}$$

and

$$\{\{u_1, u_2\}, \{u_2, u_3, u_4, \ldots, u_t\}\}$$

be optimal solutions for some $F \in \text{INPUT}_{EC}$ with $F^*_{EC} = t+l$. The corresponding solutions for $\varphi(F) = \langle T, s, b > \text{ in SS are of measure } b+1 \text{ and } b+(1+p) \text{ and } (\varphi(F))_{SS}^* = b+1$. Thus, although $EC \preceq_p \text{SS via } (\varphi, \psi)$, $(\varphi, \psi)$ does not preserve approximate solutions.

Ausiello et al. [29] considered a very interesting reducibility among optimization problems. This reducibility preserves the "structure" of the optimization problems. Or, more specifically, it preserves the number of approximate solutions at each level of approximation. The reducibility is so strong that we can easily show that some problem is not reducible to some other problem. Thus, a partial ordering among the optimization problems exists and represents the difficulty of approximation. However, we feel that this reducibility is, on the one hand, too strong to reflect our needs for an "approximate-solution-preserving" reducibility, and,
on the other hand, too weak to provide algorithms for finding approximate solutions from known approximate solutions of a harder optimization problem. Actually, if we modify the definition of $\leq_a^p$ as follows, we may get a similar but more powerful reducibility:

Assume that $R$ and $S$ are two optimization problems. We say that $R \leq_a^p S$ via $(\psi, \psi)$ if $R \leq_m^P S$ via $(\psi, \psi)$ and $\psi$ is locally one-to-one and onto, i.e., for each $u \in \text{INPUT}_R$, $\psi: \text{SOL}_S(\psi(u)) \rightarrow \text{SOL}_R(u)$ is one-to-one and onto. We say that $R \leq_a^P S$ via $(\psi, \psi)$ if $R \leq_a^P S$ and $R \leq_a^P S$ via $(\psi, \psi)$.

It is clear that $\leq_a^p$ is more powerful than $\leq_s^p$ because $\leq_s^p$ not only preserves the "structure" but also gives a method for finding the approximate solutions. However, it is not necessary to require $\psi$ to be locally one-to-one in order to preserve approximate solutions.

It is hard to directly prove that a problem $R$ is not $\leq_a^P$-reducible to another problem $S$ (although this may be easy for the stronger reducibility $\leq_s^P$). An indirect way of proving such results is to use theorem 4.4. For example, Sahni and Gonzalez [59] showed that the travelling salesman problem is not polynomial time $\varepsilon$-approximable for any $\varepsilon < 1$ unless $P = NP$; Garey and Johnson [35] showed a similar result for the graph coloring problem; and Ibarra and Kim [42] showed that the knapsack problem and the sum of subsets problem are polynomial time arbitrarily approximable. So, we have

Traveling salesman $\leq_a^P$ Knapsack, and

Graph coloring $\leq_a^P$ Sum of subsets.
The complete problems in $\Delta^p_2$ under $\leq^p_a$ may be easily constructed:

**Theorem 4.7.** There exists a function $k \in \Delta^p_2$ such that, for any $f \in \Delta^p_2$, $f \leq^p_a k$.

**Proof.** Let $A$ be a fixed NP-complete set. Define

$$k(x) = \begin{cases} 
    y, & \text{if } x = <M, w, O^n> \text{ where } M \text{ is an OTM such that } M^A(w) \text{ halts in } n \text{ steps and outputs } y. \\
    \uparrow, & \text{otherwise.}
\end{cases}$$

Then $k \in \Delta^p_2$ because the simulation of $M^A$ on $w$ for $n$ steps takes $O((|M| + |w| + n)^2)$ steps.

Assume that $f \in \Delta^p_2$. Then there exists an OTM $M^A_f$ such that $M^A_f$ computes $f$ in polynomial time. Let $p_f$ be the time complexity function for $M^A_f$. Then $f \leq^p_m k$ via $(\varphi, \psi)$, where

$$\varphi(w) = <M^A_f, w, O_f^n> \quad \text{and} \quad \psi = \lambda y[y].$$

Since $\psi$ is the identity function, it is relatively continuous. So, $f \leq^p_a k$ via $(\varphi, \psi)$. \[\]

However, this unnatural complete function is not useful for classifying problems. Besides, we are also interested in $\leq^p_a$-complete problems in NAP, PAP, or AP. The classification of known much-studied problems into these classes is very helpful. For example, Johnson [43] studied several important problems from the point of view of existing algorithms and obtained the result that the following problems are listed in the order of increasing time complexity of some natural heuristic algorithms for them:
Sum of subset (SS), Binpacking (BP), Max satisfiability (MS),
Exact cover (EC), Graph coloring (GC), Max Clique (MC).
Among them, we have already seen that \( MS \leq^P_a MC \), \( GC \not\leq^P_a SS \), and
\( GC \not\leq^P_a BP \) and we suspect that \( GC \not
\leq^P_a EC \), and \( EC \not\leq^P_a SS \) because the
natural many-one reduction functions between them don’t preserve
approximate solutions. So, this heuristic algorithm classification
matches our \( \leq^P_a \) classification. In particular, we may think GC
and MC as "hardest" problems in NAP and any problem to which GC
or MC can be \( \leq^P_a \)-reduced is considered to be in NAP; and any problem
which is \( \leq^P_a \)-reducible to SS or BP is considered to be in AP or
PAP, respectively. When this list of problems grows, the new problems
may be easier to classify.
Chapter V

Probabilistic Algorithms

5.1. Probabilistic Turing Machines

In this chapter we study polynomial time probabilistic algorithms for NP-hard problems. Probabilistic Turing machines (PTM) are good models of probabilistic algorithms. Gill [39] made a thorough study of the probabilistic Turing machine and its time complexity. We first give a short summary of his results.

A PTM is a deterministic TM equipped with a random number generator and a random state. The random number generator generates a 0 or 1 at each step. When the PTM enters the random state, the machine examines the currently generated random number (of 1 digit) and decides the new state according to it. If we compare the non-deterministic TM (NTM) with PTM, we can see the following similarity: For an NTM M and an input x, the computation of M(x) can be described as a tree and the points of branching in the tree are those points where the machine makes nondeterministic moves. Similarly, the computation of a PTM M on an input x can be described as a tree and the points of branching in the tree are those points where the machine enters the random state. In the former case, we say that the input x is accepted by the NTM M if, in the computation tree of M(x), there exists at least one path leading to acceptance of x. For the probabilistic computation, we say that the input x is accepted by the
PTM $M$ with $r \cdot 100\%$ chance of error if, in the computation tree of $M(x)$, the chance of selecting an "accepting path" is at least $1-r$.

Formally, we consider the PTM as a machine operating on two variables: one is the input $x$, the other is the sequence of random numbers generated by the random number generator. If, for a given input $x$ and a sequence $\alpha$ of random numbers of length $n$, the PTM $M$ halts in $\leq n$ steps, then we write $M(x, \alpha) \downarrow$ and $M(x, \alpha) = y$ if it is a transducer and outputs $y$. On the other hand, if $M$ with input $x$ and random string $\alpha$, $|\alpha| = n$, does not halt in $n$ steps, then we write $M(x, \alpha) \uparrow$. So, $\lambda x, \alpha [ M(x, \alpha) ]$ is a partial recursive function whose computation can be simulated by a deterministic TM.

Now, we give the definitions of the functions computed by PTM's and the time complexity of PTM's.

**Notation.** We write $(\exists x)_{m,r} R(x)$ to mean that, out of $2^m$ many possible stings over $\{0,1\}$ of length $m$, there are more than $r \cdot 2^m$ many such strings $x$ satisfying the property $R(x)$.

**Definition 5.1.** [39] The function computed by a PTM $M$ is

$$f(x) = \begin{cases} y, & \text{if } (\exists n) (\exists \alpha)_n, 1/2 \ M(x, \alpha) = y. \\ \uparrow, & \text{otherwise}. \end{cases}$$

**Definition 5.2.** [39] The time complexity function $T_M$ of a PTM $M$ which computes the function $f$ is defined as

$$T_M(x) = \begin{cases} \mu_n [(\exists \alpha)_n, 1/2 [ M(x, \alpha) = f(x) ]], & \text{if } f(x) \downarrow. \\ \infty, & \text{if } f(x) \uparrow. \end{cases}$$
Note that the time complexity of $M$ on $x$ is different from the "average time" for computing $f(x)$. This was discussed in [39].

**Definition 5.3.** A PTM $M$ is of polynomial time complexity if there is a polynomial $p$ such that $T_M(x) \leq p(|x|)$.

**Definition 5.4.** [39] A PTM $M$ computes $f$ with bounded error in polynomial time if there are an $r > 1/2$ and a polynomial $p$ such that

$$(\forall x)(\forall y) \left[f(x) = y \iff (\exists \alpha)p(|x|), r M(x, \alpha) = y\right].$$

Note that if $M$ computes $f$ with bounded error in polynomial time then, for any small $\varepsilon$, $0 < \varepsilon < 1/2$, there exists a PTM $M'$ computing $f$ in polynomial time with error $< \varepsilon$. All $M'$ does is to repeat the computation of $M$ several times so that the chance of error is decreased.

The analogue of the "P = NP?" problem for probabilistic computation is the "P = PP?" problem where PP and other classes of languages are defined in the following.

**Definition 5.5.** [39] (i) A language $L$ is in PP if there exist a PTM $M$ and a polynomial $p$ such that

$x \in L \Rightarrow (\exists \alpha)p(|x|), 1/2 M(x, \alpha) = 1$, and

$x \notin L \Rightarrow (\exists \alpha)p(|x|), 1/2 M(x, \alpha) = 0$.

(ii) A language $L$ is in BPP if there exist a PTM $M$, a polynomial $p$ and a number $r$, $1/2 < r < 1$, such that
\( x \in L \Rightarrow (\exists \alpha)_{p(|x|), r} M(x, \alpha) = 1 \), and
\( x \notin L \Rightarrow (\exists \alpha)_{p(|x|), r} M(x, \alpha) = 0 \).

(iii) A language \( L \) is in \( \text{VPP} \) (or, called \( \text{R} \) in [58]) if there exist a PTM \( M \) and a polynomial \( p \) such that
\( x \in L \Rightarrow (\exists \alpha)_{p(|x|), 1/2} M(x, \alpha) = 1 \), and
\( x \notin L \Rightarrow (\exists \alpha)_{p(|x|), 1} M(x, \alpha) = 0 \).

The definition of \( \text{VPP} \) is motivated by Rabin's result [57] that the primality problem is in \( \text{VPP} \). Rabin also mentioned that if both \( L \) and \( \overline{L} \) are in \( \text{VPP} \), then by dovetailing the computation of "\( x \in L? \)" and "\( x \notin L? \)" repeatedly we can get a correct answer (with probability of error equal to 0) with the average run time bounded by a polynomial. The relations between \( \text{PP} \), \( \text{BPP} \), \( \text{VPP} \) and \( P \), \( \text{PP} \) are as follows [39]:

\[
P \subseteq \text{VPP} \subseteq \text{BPP} \subseteq \text{PP} \subseteq \text{PSPACE}
\]

\( \text{VPP} \subseteq \text{NP} \subseteq \text{PP} \)

None of the inclusions in the above are known to be proper. We do know that \( \text{PP} \) is closed under complement and so, unless \( \text{NP} = \text{co-NP} \), \( \text{NP} \neq \text{PP} \).

5.2. A Negative Result

Whether \( \text{NP} = \text{VPP} \) or not is an interesting question. Obviously, \( \text{VPP} \neq \text{NP} \Rightarrow P \neq \text{NP} \). We tend to believe that \( \text{VPP} \neq \text{NP} \) and whether \( \text{VPP} = \text{NP} \) or not is hard to prove. Baker, Gill, and Solovay [30] proved that, in relativized cases, both \( P = \text{NP} \) and \( P \neq \text{NP} \) may happen. The same situation exists for "\( \text{VPP} = \text{NP} ? \)"
question and suggests that the direct simulation technique (to show that VPP = NP) or the straightforward diagonalization technique (to show that VPP ≠ NP) is not likely to work.

**Theorem 5.1.** [58] (i) (∃ A, recursive) \[ P^A = \text{VPP}^A \neq \text{NP}^A \].

(ii) (∃ B, recursive) \[ P^B \neq \text{VPP}^B = \text{NP}^B \].

The situation for "NP = PP?" is similar.

**Theorem 5.2.** (i) (∃ A, recursive) \[ \text{NP}^A = \text{PP}^A \].

(ii) (∃ B, recursive) \[ \text{NP}^B \neq \text{PP}^B \].

**Proof.** (i) From [30], there exists a set A such that

\[ P^A = \text{PSPACE}^A \] (just let A be PSPACE-complete). So, \[ P^A = \text{NP}^A = \text{PP}^A \].

(ii) From [30], there exists a set B such that \[ \text{NP}^B \] is not closed under complement. So, \[ \text{NP}^B \neq \text{PP}^B \].

It may be interesting to know the relationship between PP and PH, the polynomial time hierarchy [63]. However, even the relativized questions are hard to answer. For example, is there a set A such that \[ P^A = \text{NP}^A \neq \text{PP}^A \]? The yes answer to this question will mean that \[ P^A = \text{PH}^A \neq \text{PP}^A \] \[ \subseteq \text{PSPACE}^A \] which is not known yet. On the other hand, if the answer is no, then \[ P = \text{NP} \Rightarrow P = \text{PP} \] which would also be a surprising result. We consider here the relationship between PP and \( \Delta^P_2 \) and show that it depends on "NP = VPP?". In practical, \( \Delta^P_2 \subseteq PP \) would mean that all optimization problems we considered in chapter 4 can be solved by some probabilistic algorithms in polynomial time with the probability of error less than 1/2. We show that this is not the case unless \( NP = VPP \).
Definition 5.6. Let \( S = (\text{INPUT}_S, \text{SOL}_S, m_S) \) be an optimization problem as defined in chapter 4. We say that an algorithm \( A \) is a reasonable probabilistic algorithm for \( S \) if

(i) \( (\forall u \in \text{INPUT}_S) \quad \Pr \{ \text{Output}_A(u) \in \text{SOL}_S(u) \} = 1 \); 
(ii) \( (\forall u \in \text{INPUT}_S) \quad \Pr \{ m_S(\text{Output}_A(u)) = u^*_S \} > 1/2 \).

That is, \( A \) always gives an approximate solution and, with the probability greater than 1/2, \( A \) computes the optimal solutions.

Lemma 5.1. \( \leq_P^m \) preserves the membership in VPP.

Proof. Assume that \( A \leq_P^m B \) via \( f \) and there are a PTM \( M \) and a polynomial \( p \) such that

\[
\begin{align*}
x \in B & \Rightarrow (\exists \alpha)_{p(|x|),1/2} M(x, \alpha) = 1, \quad \text{and} \\
x \notin B & \Rightarrow (\exists \alpha)_{p(|x|),1} M(x, \alpha) = 0.
\end{align*}
\]

Now, consider a new PTM \( M' \) which, for given \( x \), computes \( f(x) \) and then simulate \( M \) on \( f(x) \). For this machine,

\[
\begin{align*}
\Pr \{ M'(x) = 1 \mid x \in A \} &= \Pr \{ M(f(x)) = 1 \mid f(x) \in B \} > 1/2, \\
\Pr \{ M'(x) = 0 \mid x \notin A \} &= \Pr \{ M(f(x)) = 0 \mid f(x) \notin B \} = 1.
\end{align*}
\]

Thus, \( A \in \text{VPP} \). \( \square \)

Theorem 5.3. There is no polynomial time reasonable probabilistic algorithm for the graph coloring problem unless \( \text{VPP} = \text{NP} \).

Proof. Assume, by way of contradiction, that \( A \) is a polynomial time reasonable algorithm for the graph coloring problem. Let us consider the \( k \)-graph coloring problem which is an \( \text{NP} \)-complete language recognition problem: Given a graph \( G \) and a number \( k \), is there a way of coloring the nodes of \( G \) such that no two adjacent nodes have the
same color and the total number of colors used is \( \leq k \)? We construct the following algorithm \( A' \) for the \( k \)-graph coloring problem:

Given a graph \( G \) and a number \( k \), apply \( A \) on \( G \) and, in polynomial time, get \( k' = m_{GC}(Output_A(G)) \). If \( k' \leq k \), then answer "yes" ; otherwise, answer "no" .

Then we have \( \Pr \left\{ k' \leq k \mid G \text{ can be colored by } k \text{ colors} \right\} \)

\[
= \Pr \left\{ m_{GC}(Output_A(G)) \leq G^*_GC \mid G^*_GC \leq k \right\} > 1/2 ,
\]

and

\[
\Pr \left\{ k' > k \mid G \text{ cannot be colored by } k \text{ colors} \right\} = \Pr \left\{ m_{GC}(Output_A(G)) > k \mid G^*_GC > k \right\} = 1
\]

since \( \Pr \{Output_A(G) \in SOL_{GC}(G)\} = 1 \).

So, \( k \)-graph coloring is in \( \text{VPP} \).

By lemma 5.1, \( \text{NP} \subseteq \text{VPP} \), or \( \text{NP} = \text{VPP} \). □

The above simple argument can be applied to all optimization problems whose corresponding \( k \)-language recognition problems are \( \text{NP} \)-complete ( If \( R \) is an optimization problem, then \( \text{k-R} \) is the language recognition problem defined as follows [48]

\[
\text{k-R} = \{ (u,k) \mid u \in \text{INPUT}_R, u^*_R \leq k \ (u^*_R \geq k) \}
\]

if \( R \) is a minimization (maximization) problem ). They include the traveling salesman problem, the maximum clique problem, the minimum vertex cover problem, the binpacking problem, the feedback edges problem, etc. This argument also verified the following conjecture at the abstract problem level.

**Conjecture.** \( A^P_2 \subseteq \text{PP} \Rightarrow \text{VPP} = \text{NP} \).
In the following we show the converse of this conjecture at the formal language level.

**Theorem 5.4.** \( P^{VPP} \subseteq BPP \).

**Proof.** Let \( L \) be in \( P^{VPP} \) such that an OTM \( M_1 \) with oracle \( A \in VPP \) computes \( L \) with time complexity bounded above by a polynomial \( p_1 \). Let PTM \( M_2 \) compute \( A \) with time complexity bounded above by a polynomial \( p_2 \) such that
\[
\begin{align*}
x &\in A \Rightarrow \Pr \left\{ M_2(x) = 1 \right\} > 1/2, \text{ and} \\
x &\notin A \Rightarrow \Pr \left\{ M_2(x) = 0 \right\} = 1.
\end{align*}
\]

Consider the probabilistic algorithm \( M \) for \( L \):

For given input \( x \), \( |x| = n \), simulate \( M_1 \) on \( x \). Whenever "\( y \in A? \)" is asked, simulate \( M_2 \) on \( y \) \( p_1(n) \) times. If at least once the simulation outputs 1 then answer "yes" to "\( y \in A? \)" and continue the simulation of \( M_1 \) on \( x \). If all \( p_1(n) \) simulations output 0 then answer "no" to "\( y \in A? \)" and continue.

End of algorithm.

First observe that, for input \( x \), \( |x| = n \), \( M_1 \) asked at most \( p_1(n) \) queries of the form "\( y \in A? \)" and each \( y \) is of length \( \leq p_1(n) \). So, the time for the above algorithm on \( x \), \( |x| = n \), is
\[
\begin{align*}
&= \text{computation time of } M_1 \text{ on } x \\
&+ \text{computation time of } M_2 \text{ on } y's \\
&\leq p_1(n) + p_1(n) \cdot p_2(p_1(n)).
\end{align*}
\]
and is bounded above by a polynomial over \( n \).
As for the probability of error, the probability of computing the correct \( M(x) \) is greater than or equal to the probability of getting correct answers from \( M_2 \) on all queries "\( y \in A? \)". For each query "\( y \in A? \)", we simulate \( M_2 \) on \( y \) \( p_1(n) \) times, so the probability of getting a wrong \( M_2(y) \) is \( \leq 2^{-p_1(n)} \). The probability of getting all \( M_2(y) \) correct is \( \geq (1 - 2^{-p_1(n)}) p_1(n) \) which is proved greater than or equal to \( 9/16 \) by the following lemma.

**Lemma.** If \( n \geq 2 \), then \( (1 - 2^{-m})^m \geq 9/16 \).

**Proof.** We prove this statement using induction on \( m \).

For \( m = 2 \), \( (1 - 2^{-m})^m = 9/16 \).

Assume that \( (1 - 2^{-k})^k \geq 9/16 \) for all \( k, 2 \leq k \leq m \).

Then,

\[
(1 - 2^{-m+1})^{m+1} \\
= ( (1 - 2^{-m}) + 2^{-m+1} )^m \cdot (1 - 2^{-m+1}) \\
> [ (1 - 2^{-m})^m + \binom{m}{1} (1 - 2^{-m})^{m-1} \cdot 2^{-m+1} ] \\
\cdot (1 - 2^{-m+1}) \\
= (1 - 2^{-m})^m + [ m \cdot (1 - 2^{-m})^{m-1} \cdot 2^{-m+1} \cdot (1 - 2^{-m+1}) \\
- (1 - 2^{-m})^m \cdot 2^{-m+1} ] \\
\geq 9/16 + (1 - 2^{-m})^{m-1} \cdot 2^{-m+1} \cdot [ m \cdot (1 - 2^{-m+1}) \\
- (1 - 2^{-m}) ] \\
> 9/16 . \text{ (End of the proof of lemma)}
\]

Therefore, \( L \in \text{PP} \).

**Corollary 5.4.1.** \( \text{NP} = \text{VPP} \Rightarrow A^D_2 \subseteq \text{PP} \).
6.1. Introduction

Recently much work has been done on the average case analysis of approximation algorithms. The study of the worst case analysis has shown that many practically useful algorithms are not polynomial time algorithms in the worst case despite the fact that they have been successfully applied to practical problems. It is important to know that these algorithms are, on the average, polynomial time bounded. Karp [46] has given a theoretical framework for the average case analysis and has shown that in this framework many intuitively sound algorithms are indeed "good" algorithms.

In this chapter we study the average case analysis of polynomial time approximation from an abstract point of view. We consider the problems only as formal languages contained in $\Sigma^*$ for some fixed alphabet $\Sigma$. A simplified view of the approximation is also considered. For each problem $S$ which is not in $P$, a polynomial time approximation algorithm $A$ for $S$ works as following:

1. There exists a polynomial $p$ such that, for any $x \in \Sigma^*$, $A(x)$ halts in $\leq p(|x|)$ steps and $A(x) = 1$ (accepts $x$), $0$ (rejects $x$), or $?$ (does not know the answer).

2. For every $x \in \Sigma^*$, $A(x) = 1 \Rightarrow x \in S$, and $A(x) = 0 \Rightarrow x \notin S$. 

125
The set \( \{ x \in \Sigma^* \mid A(x) \neq ? \} \) is called the "correct part" of \( A \). Obviously, the size of the correct part of an algorithm \( A \) determines how good it is.

We first consider the infinitely often (i.o.) speedable and almost everywhere (a.e.) complex questions, that is, questions which ask whether there is an algorithm \( A \) for \( S \) such that the correct part is infinite or not. Then we discuss, under different definitions of the size of a set, how large the correct part can be.

We know that a finite set is always polynomial time computable because we can determine the membership of an input in a constant number of steps by using the table look-up technique. Therefore, for any polynomial time approximation algorithm \( A \) for a set \( S \notin P \), the correct part of \( A \) is co-infinite. Actually, Lynch [51] showed that, for every \( S \notin P \), there exists a fixed "incorrect part".

**Theorem 6.1.** [51] If \( S \) is a recursive set with \( S \notin P \), then there exists an infinite recursive set \( X \) such that

\[
(\forall p, \text{ a polynomial}) \ (\forall M, \text{ a DTM}) \ \\
[ M \text{ computes } \chi_S \Rightarrow T_M(x) \geq p(|x|) \text{ a.e. on } X ]
\]

where \( T_M(x) \) is the number of steps \( M \) takes to compute \( \chi_S(x) \), and a.e. is the abbreviation of "for all but finitely many".

The set \( X \) is called a polynomial complexity core of \( S \).

It is clear that for any polynomial time approximation algorithm \( A \) for \( S \), the intersection of \( X \) and the correct part of \( A \) is finite. The size of a polynomial complexity core of \( S \) thus gives
a lower bound on the "incorrect part" of any polynomial time approximation algorithm \( A \) for \( S \).

6.2. P-productive Sets and P-immune Sets

In recursive function theory the concept of productivity has been used to study the speedability of nonrecursive sets. In particular, Blum and Marques [33] showed that a recursively enumerable set is "effectively speedable" if and only if it is "subcreative". In the following we define the notion of polynomial time productivity and show that this notion is, unfortunately, not useful at the level of \( \text{EXPTIME} = \text{DTIME}(2^{\text{linear}}) \).

We first make two basic assumptions:

1. There is an enumeration of polynomial time computable Turing machines. Call them \( M_i \), with the program encoded into a string \( \widetilde{M}_i \) and time complexity \( p_i = \lambda x [x^i + i] \).

2. This enumeration is efficient, i.e., there exist two polynomial time computable functions \( \varphi_1 \) and \( \varphi_2 \) and two polynomials \( q_1 \) and \( q_2 \) such that \( \varphi_1(i) = \widetilde{M}_i \), \( \varphi_2(\widetilde{M}_i) = i \) and \( |i| \leq q_1(|\widetilde{M}_i|) \), \( |\widetilde{M}_i| \leq q_2(|i|) \).

The assumption 1 has been used in [30]. The assumption 2 is also essential for the lemma 1 of [30]. It means that, when \( <i, x> \) is an input, we can simulate \( M_i \) on \( x \) using not more than polynomially bounded time.

Let \( P_i \) be the language recognized by \( M_i \).
Definition 6.1. A recursive set $A$ is said to be $p$-productive if there is a polynomial time computable function $f$ (called the $p$-productive function for $A$) such that whenever $P_i \subseteq A$ we have $f(i) \in A - P_i$.

Theorem 6.2. There exists a $p$-productive set in $\text{DTIME}(2^{2^{\text{linear}}})$.

Proof. Let $A = \{ \tilde{M}_1^w | M_1$ accepts $\tilde{M}_1^w$ in $\leq 2^{2^{\text{linear}}}|\tilde{M}_1^w|$ steps $\}$. We first observe that $A \in \text{DTIME}(2^{2^{\text{linear}}})$ because, for any input of the form $\tilde{M}_1^w$, we can efficiently simulate $M_1$ on $\tilde{M}_1^w$ for $2^{2^{\text{linear}}}|\tilde{M}_1^w|$ steps using only $O(|\tilde{M}_1^w|) \cdot 2^{2^{\text{linear}}}|\tilde{M}_1^w|$ steps.

Thus, $\overline{A}$ is also in $\text{DTIME}(2^{2^{\text{linear}}})$.

We claim that $\overline{A}$ is $p$-productive.

First define $f(i) = \tilde{M}_1^{0^m}$ where $m$ is a number greater than 0 such that $f(i)$ is padded with zeroes so that its length becomes $3 \cdot q_1(|\tilde{M}_1|) + 3$. It is clear that $f$ is polynomial time computable by assumption 2. Now, if $P_i \subseteq \overline{A}$ we show that $f(i) \in \overline{A} - P_i$ in the following.

(1) Suppose, by way of contradiction, $f(i) \in A$, then $M_i$ accepts $f(i) = \tilde{M}_1^{0^m}$ in $\leq 2^{2^{\text{linear}}}|\tilde{M}_1^{0^m}|$ steps. Thus, $f(i) \in P_i \subseteq A$. A contradiction. I.e., $f(i) \notin A$.

(2) Suppose, by way of contradiction, $f(i) \in P_i$. Then $M_i$ accepts $f(i)$ in $\leq p_i(|f(i)|)$ steps. By the following lemma, $p_i(|f(i)|) \leq 2^{2^{\text{linear}}}|f(i)|$. Then, by the definition of $A$, $f(i) \in A$. 

This is a contradiction and so we have shown that \( f(i) \notin p_i \).

It is left to prove the following lemma.

**Lemma.** If \( n \geq 3 \cdot q_1(\mid \tilde{M}_1 \mid) + 3 \), then \( p_i(n) \leq 2^{2n} \).

**Proof.**

\[
p_i(n) = n^i + i \leq 2^{ni} + i \leq 2^{ni} + 2^i \leq 2^{ni} \cdot 2^i = 2^i(n+1) \leq 2^{2\mid i\mid + 1 \cdot (n+1)}.
\]

For \( n \geq 3 \cdot q_1(\mid \tilde{M}_1 \mid) + 3 \), we have

\[
2^n \geq 2^{q_1(\mid \tilde{M}_1 \mid) + 1} \cdot 2^{2n/3} \geq 2^\mid i\mid + 1 \cdot 2^{2n/3} \geq 2^\mid i\mid + 1 \cdot (n+1)
\]

since \( n \geq 3 \Rightarrow 2^{2n/3} \geq n+1 \).

So, \( 2^n \geq p_i(n) \), if \( n \geq 3 \cdot q_1(\mid \tilde{M}_1 \mid) + 3 \). \(

**Theorem 6.3.** There is no \( p \)-productive set in \( \text{DTIME}(2^{\text{poly}}) \).

**Proof.** Assume, by way of contradiction, that \( A \) is \( p \)-productive and can be computed by a deterministic Turing machine (DTM) \( M_A \) with time complexity \( 2^{p_A(n)} \) for some polynomial \( p_A \). Let \( f \) be the \( p \)-productive function for \( A \) with time complexity \( p_f \), a polynomial.

Now, for each integer \( i \), construct a DTM \( M_{i}' \) as follows:

For given input \( x \), \( M_{i}' \) simulates \( M_A \) on \( x \) for \( |x|^i \) steps and accepts \( x \) only if \( M_A \) accepts \( x \) in \( |x|^i \) steps. It is clear that the machine \( M_{i}' \) is of polynomially bounded time complexity.

Since the construction of \( M_{i}' \) is uniform, there is a recursive function \( \sigma \) such that \( M_{i}' = M_{\sigma(i)} \) where \( \{M_j\} \) is the standard enumeration of polynomial time DTM in assumption 1.
\( \sigma \) is polynomial time computable because the map \( \lambda i [M_i^* \cdot ] \) can be easily computed in polynomial time by attaching a "clock machine" to \( M_A \). So, we may assume the existence of a polynomial \( q \) such that 
\[ q(|i|) \geq |\sigma(i)| \quad \text{and} \quad |\sigma(i)| \geq |i|. \]

It is clear from the description of \( M_i^* \) that \( P_{\sigma(i)} \subseteq A. \)
So, \( f(\sigma(i)) \in A = P_{\sigma(i)}. \)

Consider \( i \) so large that
\[ 2^{|i|} \geq p_A \circ p_f \circ q(|i|). \]
Then,
\[ f(\sigma(i)) \in A \]
\[ \Rightarrow M_A \text{ accepts } f(\sigma(i)) \text{ in } \leq 2^{P_A(|f(\sigma(i))|)} \text{ steps} \]
and
\[ P_A(|f(\sigma(i))|) \]
\[ \leq p_A \circ p_f(|\sigma(i)|) \]
\[ \leq p_A \circ p_f \circ q(|i|) \]
\[ \leq 2^{|i|} \leq i. \]
Thus, \( M_A \) accepts \( f(\sigma(i)) \) in \( \leq 2^i \) steps, or \( \leq |f(\sigma(i))|^i \) steps.
So, \( f(\sigma(i)) \) is accepted by \( M_{\sigma(i)} \) and is in \( P_{\sigma(i)}. \) This is a contradiction and we have shown that such an \( A \) does not exist. []

So, the concept of \( p \)-productivity is not very useful in studying the structure of sets in \( \text{EXPTIME}. \) On the other hand, \( p \)-productivity is not the only tool we have. It is well-known that a recursively enumerable set is \( m \)-complete if and only if its complement is productive. Also, Gill and Morris [40] showed that subcreativity is equivalent to "S-complete". So, we may well consider the relationship between completeness and speedability. Actually, L. Berman [31]
showed that if $S$ is any set which is $\leq_{m}^{P}$-complete for a deterministic time class $T(n)$ then $S$ has effective i.o. speedup from $T(n^r)$ to polynomial time for some $r > 0$.

What about other sets which are not $\leq_{m}^{P}$-complete? What about $\leq_{tt}^{P}$-complete sets? We try to answer these questions in the following. First we define $p$-immune sets as the analogue of the immune sets at the polynomial time level.

**Definition 6.2.** An infinite set $A$ is said to be $p$-immune if, for every subset $B \subseteq A$, $B \in P \Rightarrow B$ is finite.

We first show that there exists a set $S$ in EXPTIME such that both the set $S$ and its complement are $p$-immune. In other words, $\Sigma^*$, the set of all input strings, is a polynomial complexity core of $S$.

**Definition 6.3.** [41] We say that a set $A$ is $p$-sparse if there exists a polynomial $p$ such that, for any $n$,

$$\left|\{x \in \Sigma^* \mid x \in A \text{ and } |x| \leq n\}\right| \leq p(n).$$

**Lemma 6.1.** [41] There exists a set $A_0$ in EXPTIME such that

(i) $A_0$ is not $p$-sparse, and

(ii) if $A_0 \leq_{m}^{P} B$ via $\rho$, then $\rho$ is a one-to-one mapping a.e.

**Theorem 6.4.** Both the set $A_0$ in lemma 6.1 and its complement $\overline{A_0}$ are $p$-immune.

**Proof.** First note that $A_0$ must be infinite and co-infinite because otherwise $A_0$ would be in $P$ and can be $\leq_{m}^{P}$-reducible to any set
and this contradicts (ii) of lemma 6.1.

Assume, by way of contradiction, that $A_0$ has an infinite subset $C$ which is in $P$. Now, let $B$ be a $\leq_P^m$-complete set in EXPTIME and $A \leq_P^m B$ via $\rho$. Then, the function $\rho'$ defined by

$$
\rho'(x) = \begin{cases} 
\rho(x), & \text{if } x \notin C, \\
x_0, & \text{if } x \in C,
\end{cases}
$$

where $x_0$ is a fixed number in $B$, $\leq_P^m$-reduces $A_0$ to $B$ but is not one-to-one a.e. This contradicts (ii) of lemma 6.1 and hence such an infinite set $C$ does not exist. Or, $A_0$ is $p$-immune.

A similar proof can show that $\overline{A}_0$ is $p$-immune. \[\]

We can also construct a $p$-immune set using the technique of constructing Post's simple set. We use this technique to show the following theorem.

We assume that all numbers are written in binary form as strings in $\{0, 1\}^*$. 

**Theorem 6.5.** There exists a $p$-immune set which is $\leq_P^t$-complete in EXPTIME.

**Proof.** We first construct a set $S$ in EXPTIME such that $\overline{S}$ is $p$-immune and, for any $n$,

$$| S \cap \{ x \in \sum^* \mid x \leq 2^n \} | \leq \log_2 n + 1$$

by the following pidgen-PL/I algorithm $M_S$: 

\[\]
Get input $x$;

\[
\begin{align*}
&\text{do } i = 0 \text{ to } \log_2|x|; \\
&\text{do } j = 2^{i^2} \text{ to } x; \\
&\text{Simulate } M_i \text{ on } j \text{ for } 2^{|j|} \text{ steps.} \\
&\text{If } j \text{ is accepted and } j = x \\
&\text{then accept } x \text{ and halt;} \\
&\text{else if } j \text{ is accepted and } j < x \\
&\text{then go to OUT;}
\end{align*}
\]

OUT: end;

reject $x$ and halt;

We now check the following assertions ((1) - (4));

(1) $S \in \text{EXPTIME}:$

If $|x| = n$, then $x < 2^{n+1}$ and so the execution time of $M_S$ on $x$ is $\leq c_1 + (\log_2 n + 1) \cdot x \cdot (c_2 \cdot 2^n)$, for some constants $c_1$ and $c_2$, which is $= O(2^{4n})$.

(2) $\overline{S}$ is infinite:

For any $x \in S$, from the algorithm $M_S$, we know that there exists exactly one $i$ such that

(a) $i \leq \log_2|x|$

(b) the computation of $M_i$ on $x$ is simulated by $M_S$ on $x$ and it halts in $2^{|x|}$ steps, and

(c) $(\forall j, 2^i \leq j < x) [M_i \text{ does not accept } j \text{ in } \leq 2^{|j|} \text{ steps}].$

The reason is that if $M_i$ accepts some $j$, $2^i \leq j < x$, in $\leq 2^{|j|}$ steps then the execution flows to out, and $M_i$ is no
longer to be simulated.

Let \( \varphi : S \rightarrow N \) be defined by \( \varphi(x) = i \) described above. We claim that \( \varphi \) is one-to-one.

Assume, by way of contradiction, that \( x < y \) and \( \varphi(x) = \varphi(y) = i \). Then, by (a), (b) and \( \varphi(x) = i \), we have \( i \leq \log_2|x| \) and \( M_i \) accepts \( x \) in \( \leq 2^{|x|} \) steps. However, by (c) and \( \varphi(y) = i \), we have that \( M_i \) does not accept \( x \) in \( \leq 2^{|x|} \) steps. This is a contradiction. So, \( \varphi \) is one-to-one.

But the fact that \( \varphi \) is one-to-one means that
\[
|S \cap \{x \mid |x| \leq n\}| \leq \log_2 n.
\]
Hence \( \overline{S} \) is infinite.

(3) \( \overline{S} \) has no infinite subset in \( P \):

By the way of contradiction, assume that \( P_k \subseteq \overline{S} \) and \( P_k \) is infinite.

Since \( 2^n > p_k(n) \) a.e., we know that the set
\[
\{x \in \Sigma^* \mid |x| > 2^k \text{ and } M_k \text{ accepts } x \text{ in } \leq 2^{|x|} \text{ steps}\}
\]
is infinite. Let \( x_k \) be the smallest element in this set. Then \( x_k \in P_k \subseteq \overline{S} \).

Consider the algorithm \( M_S \) on \( x_k \). Since \( M_S \) will reject \( x_k \), \( M_S \) will go through the simulation of \( M_i \) on \( j \),
\[
2^i \leq j \leq x_k, \text{ for every } i = 0, 1, \ldots, \log_2|x_k|.
\]
So, \( M_S \) will simulate \( M_k \) on \( j \)'s, \( 2^k \leq j \leq x_k \). But \( x_k \) is the smallest number in \( \{2^k, \ldots, x_k\} \) which \( M_k \) will accept in \( \leq 2^{|x|} \) steps. So, \( M_S \) will eventually simulate \( M_k \) on \( x_k \) for \( 2^{|x_k|} \) steps and accepts \( x_k \). This contradicts the fact that
\( x_k \notin S \). So, we have shown that \( \overline{S} \) has no infinite subset in \( P \).

(4) \[ |S \cap \{ x \in \Sigma^* \mid x \leq 2^n \} | \leq \log_2 n + 1; \]

This has been shown in (2).

Now we construct finite sets \( S_n = \{ y_n, y_n+1, \ldots, y_n+k_n-1 \} \)
where \( k_n = \lceil \log_2 n \rceil \) and \( y_n = nk_n - 2^{k_n} - k_n + 1 \).

First observe the following facts (5) & (6):

(5) The function \( \lambda n[y_n] \) is polynomial time computable because \( \lambda n[k_n] \) can be computed by binary searching.

(6) For all but finitely many \( n \), \( S_n \cap \overline{S} \neq \emptyset \).

Since \( |S \cap \{ x \in \Sigma^* \mid x \leq 2^n \} | \leq \log_2 n + 1 \), we need only to check that, for almost all \( n \), \( y_n + k_n - 1 < 2^{n/2} \). (So, \( |S \cap S_n| \leq |S \cap \{ x \in \Sigma^* \mid x < 2^{n/2} \}| < \log_2 n \leq k_n \) and hence \( |S \cap S_n| \geq 1 \).

But \( y_n + k_n - 1 = n \cdot k_n - 2^{k_n} \leq n^2 \leq 2^{n/2} \)
if \( n \) is large enough.

Now we assume that the set \( A \) is \( \leq^p_{m} \)-complete in EXPTIME.
Let \( S^* = S \cup \{ S_n \mid n \in A \} \). We claim that \( S^* \) is \( m \)-immune and \( \leq^p_{tt} \)-complete in EXPTIME (5) & (9):

(7) \( S^* \in \text{EXPTIME} \):

Given input \( x \), first perform the binary search over \( k \)
such that \( y_k \leq x < y_{k+1} \) (From (5), this can be done in \( p(|x|) \) steps for some polynomial \( p \)). Then test whether \( k \in A \). If yes, then accept \( x \); otherwise, accept \( x \)
if and only if \( x \) is in \( S \). (Both \( A \) and \( S \) are in EXPTIME).

(8) \( S^* \) is \( \leq^p_{tt} \)-complete in EXPTIME (and so is \( \overline{S^*} \)).

We need only to show that \( A \leq^p_{tt} S^* \).
Note that for $n$ large enough, $\bar{S} \cap S_n \neq \emptyset$, so,
$$(\forall n)[n \in A \iff S_n \subseteq S^*]$$
where $\forall n$ is the abbreviation of "for all but finitely many $n$".
That is, $A \leq_{tt}^p S^*$ via the following $tt$-function $<f, g>$:

- $f(n) = <y_n, y_{n+1}, \ldots, y_{n+k_n-1}>$,
- $g(<n, m_1, m_2, \ldots, m_{k_n}>)$ is true if and only if
  $m_1 = m_2 = \ldots = m_{k_n}$ is true, for large enough $n$; and
- $f$ and $g$ are trivial on small $n$.

(9) $\bar{S}^*$ is $p$-immune.

Since $\bar{A}$ is infinite and $(\forall n)[S_n \cap \bar{S} \neq \emptyset]$, $S^*$ must be infinite. Now if $B \in P$ is a subset of $S^*$, then $B \subseteq S$ and $B$ must be finite. \[\]

Combining theorem 6.5 and L. Berman's result in [31], we can only conclude that, in EXPTIME, $\leq_m^p$-complete sets are always i.o. speedable, and this may not be true for $\leq_{tt}^p$-complete sets. In [40], effective speedable sets are shown to be equivalent to $S$-complete sets and $\leq_S$ is incomparable with $\leq_{tt}$. An interesting open question is whether there exists a polynomial time reducibility $\leq_S^p$ as an analogue of $\leq_S$ which satisfies similar properties.

As a summary, the sets in EXPTIME may be classified in the following spectrum according to their "richness" of possession of easy subsets.
\[ \mathcal{B}_0 = \mathcal{P}. \]

\[ \mathcal{B}_1 = \{ A \in \text{EXPTIME} \mid A \text{ is } p\text{-immune} \}. \]

\[ \mathcal{B}_2 = \{ A \in \text{EXPTIME} \mid (\exists B, C) \left[ B \text{ and } C \text{ are infinite, } B \in \mathcal{P}, \right. \]
\[ \left. C \text{ is } p\text{-immune, and } A = B \cup C \right\}. \]

\[ \mathcal{B}_3 = \{ A \in \text{EXPTIME} \mid (\forall B \in \mathcal{P}) \left[ B \subseteq A \Rightarrow (\exists C \in \mathcal{P}) \left[ C \subseteq A - B \right. \right. \]
\[ \left. \text{ and } C \text{ is infinite }\right] \}, \text{ but there is no efficient (polyno-} \]
\[ \text{mial time) way of finding } C \text{ from } A \text{ and } B \}. \]

\[ \mathcal{B}_4 = \{ A \in \text{EXPTIME} \mid A \text{ is } p\text{-productive} \}. \]

We have seen that actually \( \mathcal{B}_4 \) is empty. \( \mathcal{B}_0 - \mathcal{B}_3 \) form a mutually exclusive and exhaustive classification, and they are all nonempty. The \( \lesssim_{m}^{p} \)-complete sets in \text{EXPTIME} are contained in class \( \mathcal{B}_3 \).

6.3. Density and Complexity

We have just seen that the size of the cores of sets in \text{EXPTIME} could be as large as the set \( \Sigma^* \) of all inputs but a \( \lesssim_{m}^{p} \)-complete set must have an infinite "easy" part. How large can this "easy" part be? Is there an upper bound for it? We now discuss these questions.

**Definition 6.4.** [50, 53] The density of a set \( S \subseteq \mathbb{N} \) is less than (greater than) \( r \), for \( 0 \leq r \leq 1 \), if

\[
\text{dens}_n(S) \equiv \frac{|S \cap \{0, 1, \ldots, n\}|}{n+1} < r \ (> r) \ a.e.
\]

This definition basically assumes the uniform distribution over any initial segment of integers. We may compare it with Karp's [46]
more practical definition.

Definition 6.5. [46] Let $S_n$ be a distribution over $x \in \{0,1\}^n$ such that

$\sum_{n=0}^{\infty} \Pr \left\{ x \cap \{0,1\}^n \right\} S_n < \infty$

and $X = Y$ a.e. if $(X-Y) \cup (Y-X) = \emptyset$ a.e.

The following theorem shows that, under the uniform distribution, the notion $X = \emptyset$ a.e. is stronger than the notion $\text{dens}_n(X) = 0$ a.e., and p-sparseness is even stronger. So, these definitions can be used to study the size of sets in more detail.

Theorem 6.6. In the following, (i) $\Rightarrow$ (ii) $\Rightarrow$ (iii), but (ii) $\not\Rightarrow$ (i), and (iii) $\not\Rightarrow$ (ii).

(i) $A$ is p-sparse.

(ii) $A = \emptyset$ a.e. under uniform distribution.

(iii) $(\forall \varepsilon > 0) \left\lfloor \text{dens}_n(A) < \varepsilon \right\rfloor$ a.e.

Proof. (i) $\Rightarrow$ (ii). Assume that $A$ is p-sparse and $q$ is a polynomial such that $|A \cap \{0,1\}^n| \leq q(n)$.

Now

$\sum_{n=0}^{\infty} \Pr \left\{ A \cap \{0,1\}^n \right\}$ uniform

$\leq \sum_{n=0}^{\infty} q(n)/2^n < \infty$

So, $A = \emptyset$ a.e. under uniform distribution.

(ii) $\not\Rightarrow$ (i). Let $A$ be a set $\subseteq \{0,1\}^*$ such that

$|A \cap \{0,1\}^n| = 2^{n/2}$. Then $A$ is not p-sparse but
\[ \sum_{n=0}^{\infty} \Pr\{A \cap \{0,1\}^n\}_{\text{uniform}} \leq \sum_{n=0}^{\infty} 2^{-n/2} < \infty. \]

So, \( A = \emptyset \) a.e. under uniform distribution.

(ii) \( \Rightarrow \) (iii). Assume that \( A = \emptyset \) a.e. under uniform distribution. Let \( a_n = |A \cap \{0,1\}^n| \). then we have

\[ \sum_{n=0}^{\infty} a_n/2^n = c < \infty. \]

All we need is to show that

\[(\forall \varepsilon > 0) \ \text{dens}_n(A) < \varepsilon \ \text{a.e.} \]

that is,

\[(\forall \varepsilon > 0) \ \frac{1}{2^{n+1}} \sum_{i=0}^{n} a_i < \varepsilon \ \text{a.e.} \]

or,

\[ \frac{1}{2^n} \sum_{i=0}^{n} a_i \to 0 \text{ as } n \to \infty. \]

Now,

\[ \sum_{n=0}^{\infty} a_n/2^n = c < \infty, \]

so,

\[(\forall \varepsilon > 0) \ (\exists N_1) \left[ c/2^{N_1} < \varepsilon \right] \text{ and } \]

\[(\exists N_2) \left[ \sum_{i=N_2}^{\infty} (a_i/2^i) < \varepsilon \right]. \]

Take \( N_0 = N_1 + N_2 \), then, for \( n > N_0 \), we have

\[ 2^{-n} \cdot \sum_{i=0}^{n} a_i \]

\[ \leq 2^{-n} \cdot \sum_{i=0}^{N_2} a_i + 2^{-n} \cdot \sum_{i=N_2}^{n} a_i \]

\[ \leq 2^{-1} c + \sum_{i=N_2}^{n} \left( a_i/2^i \right) < \varepsilon + \varepsilon = 2\varepsilon. \]
So, $2^{-n} \sum_{i=0}^{n} a_i \rightarrow 0$ as $n \rightarrow \infty$.

(iii) $\not\rightarrow$ (ii). Let $a_n = 2^n/n$. Then for the set $A$ which satisfies the property $|A \cap \{0,1\}^n| = a_n$ for $n = 1, 2, \ldots$, we have

$$\sum_{n=1}^{\infty} \frac{a_n}{2^n} = \sum_{n=1}^{\infty} n^{-1} = \infty,$$

and hence $A \notin \emptyset$ a.e. However, we claim that

$$\text{dens}_n(A) = 2^{-(n+1)} \sum_{i=1}^{n} a_i \overset{\text{def}}{=} b_n \leq 4/n,$$

(hence $(\forall \varepsilon > 0 \exists N \in \mathbb{N}) [ b_n < \varepsilon \text{ a.e.} ]$).

(proof of claim). We prove it by induction on $n$.

For $n = 1$, $b_1 = (1/4) 2 = 1/2 < 4$.

Now, assume that $b_k \leq 4/k$, then

$$b_{k+1} = 2^{-(k+2)} \sum_{i=1}^{k+1} a_i \leq (1/2) \left( 2^{-(k+1)} \sum_{i=1}^{k} a_i + 2^{-(k+1)} a_{k+1} \right) = (1/2) \left( b_k + 1/(k+1) \right) \leq (1/2) \left( 4/k + 1/(k+1) \right) < 4/(k+1).$$

We now use these definitions to study the size of the polynomial complexity core of sets in \textsc{EXPTIME}.

\textbf{Theorem 6.7.} If $A$ is \textsc{PSPACE}-complete in \textsc{EXPTIME}, then for any $B, C \in \textsc{P}$,

$$[B \subseteq A, C \subseteq \overline{A}] \Rightarrow \overline{B \cup C} \text{ is not } \textsc{P}-\text{sparse}.$$

(Meyer and Paterson [54] called the class of the sets satisfying the above condition \textsc{APT}).
Proof. This is a simple corollary of lemma 6.1 (due to Hartmanis and L. Berman [41]). Actually, it is proved in [41] that lemma 6.1 implies that \( \leq^P_m \)-complete sets in \( \text{EXPTIME} \) cannot be \( p \)-sparse. It can be easily strengthened to show this theorem. \( \square \)

The above theorem says that the "incorrect" part of any polynomial time algorithm for a \( \leq^P_m \)-complete set in \( \text{EXPTIME} \) must be greater than any \( p \)-sparse set. Can it be of density 0? We give the positive answer in the next theorem.

**Theorem 6.8.** There exists a set \( A \) which is \( \leq^P_m \)-complete in \( \text{EXPTIME} \) such that

\[
(\exists \, B, C \in P) \left[ B \subseteq A, \, C \subseteq \overline{A}, \text{ and } \forall \varepsilon > 0 \left[ \text{dens}_n (B \cup C) > 1 - \varepsilon \text{ a.e.} \right] \right].
\]

**Proof.** Let \( X \) be a \( \leq^P_m \)-complete set in \( \text{EXPTIME} \). Let

\[
A = \left\{ x \in \Sigma^* \mid (\exists y \in X) \ y^2 = x \right\}.
\]

Then, obviously, \( A \) is \( \leq^P_m \)-complete in \( \text{EXPTIME} \). To see that \( A \) satisfies the above condition, just let \( B = \emptyset, \ C = \left\{ x \in \Sigma^* \mid (\exists y \in \Sigma^*) \ y^2 \neq x \right\} \) and observe that

\[
\text{dens}_n \left\{ x \in \Sigma^* \mid (\exists y \in \Sigma^*) \ y^2 = x \right\} < \varepsilon \quad \text{a.e.} \ \square
\]

Actually, the "incorrect" part, or, the polynomial complexity core of a \( \leq^P_m \)-complete set in \( \text{EXPTIME} \) could be of any size.

**Theorem 6.9.** There exists a set \( A \) which is \( \leq^P_m \)-complete in \( \text{EXPTIME} \) such that, for any sets \( B, \ C \) in \( P \),

\[
[B \subseteq A \ \& \ C \subseteq \overline{A}] \Rightarrow (\forall \varepsilon > 0) \left[ \text{dens}_n (B \cup C) < \varepsilon \text{ a.e.} \right].
\]

**Proof.** First note that, from the proof of theorem 6.5, there exists
a set $S$ in \textit{EXPTIME} such that $\overline{S}$ is $p$-immune and $S$ is $p$-sparse. Thus, for any $B, C \in P$, if $B \subseteq A$, $C \subseteq \overline{A}$, then $B \cup C$ is $p$-sparse and hence $\text{dens}_n(B \cup C) < \varepsilon$ a.e. for all $\varepsilon > 0$.

All we need to do is to modify $S$ to get a $\leq^P_m$-complete set $A$ in \textit{EXPTIME} such that density of $A$ is small and $\overline{A}$ is hard to approximate. This can be done easily by the following: Let $Y = \{x \mid x \text{ is a perfect square}\}$, and $f : N \rightarrow Y$ be a one-to-one, onto, increasing function between $N$ and $Y$. (That is, $f(n)$ is the $n$-th smallest element in $Y$). Then both $f$ and $f^{-1}$ are polynomial time computable. Now, for a $\leq^P_m$-complete set $X$ in \textit{EXPTIME}, let $A = \{x \in Y \mid (\exists y \in X) \ y^2 = x\}$ U $f(S)$.

It is easily observed that $A$ is $\leq^P_m$-complete in \textit{EXPTIME} and that, for any $\varepsilon > 0$, $\text{dens}_n(A) < \varepsilon$ a.e. So, all we have left is to show that if $C \in P$ and $C \subseteq \overline{A}$, then $\text{dens}_n(C) < \varepsilon$ a.e.

Assume that $C \in P$ and $C \subseteq \overline{A}$. Then $D = C \cap Y$ is in $P$. Now, $f^{-1}(D)$ is also in $P$ and $f^{-1}(D) \subseteq \overline{A}$. So, $f^{-1}(D)$ is finite and hence $D$ is finite. Therefore,

$$\text{dens}_n(C) \leq \text{dens}_n(D) + \text{dens}_n(Y) < \varepsilon \text{ a.e.}$$

Theorem 6.10. For any $r \in Q$, $0 < r < 1$, there exists a set $A$ which is $\leq^P_m$-complete in \textit{EXPTIME} such that

(i) $(\forall B, C \in P) \left[ \left[ B \subseteq A \text{ and } C \subseteq \overline{A} \right] \Rightarrow (\forall \varepsilon > 0) \left[ \text{dens}_n(B \cup C) < r + \varepsilon \text{ a.e.} \right] \right]$

(ii) $(\exists B, C \in P) \left[ B \subseteq A, C \subseteq \overline{A} \Rightarrow (\forall \varepsilon > 0) \left[ \text{dens}_n(B \cup C) > r - \varepsilon \text{ a.e.} \right] \right]$
Proof. Intuitively, if \( r = a/b \), we consider the intervals of \( b \) consecutive integers. For each such interval, choose \( a \) numbers in \( \mathbb{A} \), and \( b-a \) numbers in \( A \) or in \( \mathbb{A} \) so that \( A \) is complex enough. We formally prove it in the following.

Let \( A_1 \) be the set found in theorem 6.9. That is, \( A_1 \) is \( \leq_{\text{m}}^{\text{D}} \) complete in \( \text{EXPTIME} \) and if \( B, C \subseteq P \) and \( B \subseteq A \) and \( C \subseteq \mathbb{A} \), then
\[
\text{dens}_n (B \cup C) < \varepsilon \quad \text{a.e. for all } \varepsilon > 0.
\]

We define
\[
A = \{ x \mid \lfloor x/b \rfloor \in A_1 \land (\forall y) \left[ 0 \leq y < a \Rightarrow x \not\equiv y \mod b \right] \}.
\]
Then, it is easy to see that \( A \) is \( \leq_{\text{m}}^{\text{D}} \) complete in \( \text{EXPTIME} \)
\( A_1 \leq_{\text{m}}^{\text{D}} A \) via \( \lambda x \left[ bx + a \right] \). We check the conditions (i) and (ii) in the following.

(i). Since \( A = \{ b \cdot x + y \mid a \leq y < b, x \in A_1 \} \), we have that, for any \( \varepsilon > 0 \),
\[
\text{dens}_n (A) = (b - a) \cdot \text{dens}_n (A_1) < (b - a) \cdot \varepsilon \quad \text{a.e.}
\]
And so, for \( n \) large enough, we have
\[
\text{dens}_n (A) < (b - a) \cdot \varepsilon + b/n.
\]
Thus,
\[
\text{dens}_n (A) < \varepsilon \quad \text{a.e. for all } \varepsilon > 0.
\]

Now, if \( C \subseteq P \) and \( C \subseteq \mathbb{A} \), let \( D = C \cap \{ x \mid (\exists y) \left[ 0 \leq y < a \land x \equiv y \mod b \right] \} \) and \( E = C - D \). Then
\[
\text{dens}_n (D) < \text{dens}_n (\{ b \cdot x + y \mid 0 \leq y < a \}) < r + \varepsilon \quad \text{a.e. for all } \varepsilon > 0.
\]

The only thing left to show is that
\[
(\forall \varepsilon > 0) [ \text{dens}_n (E) < \varepsilon \quad \text{a.e.} ].
\]
For each \( y, a \leq y \leq b-1 \), let \( E_y = \{ x \in E \mid x \equiv y \mod b \} \) and \( f_y(x) = \lfloor (x-y)/b \rfloor \). Then \( f_y \) is a polynomial time computable one-to-one mapping on \( E_y \) and \( f(E_y) \subseteq \mathbb{A}_1 \).
Since \( C \subseteq P \), we know that \( E_y \in P \), and that \( f(E_y) \subseteq \mathbb{A}_1 \) is in \( P \).
So, for all \( \varepsilon > 0 \), we have

\[
\text{dens}_{b,n}(E_y) = (1/b) \cdot \text{dens}_n(f_y(E_y)) < \varepsilon/b \text{ a.e.}
\]

Thus, \( E = \bigcup_{y=a}^{b-1} E_y \) implies that \( (\forall \varepsilon > 0) \left[ \text{dens}_n(E) < \varepsilon \text{ a.e.} \right] \).

So, \( (\forall \varepsilon > 0) \left[ \text{dens}_n(C) = \text{dens}_n(D) + \text{dens}_n(E) < r + \varepsilon \text{ a.e.} \right] \) and \( (\forall \varepsilon > 0) \left[ \text{dens}_n(A) < \varepsilon \text{ a.e.} \right] \).

Hence, condition (i) is satisfied by \( A \).

(ii). Let \( B = \emptyset \) and \( C = \{ b x + y \mid 0 \leq y \leq a \} \). Then \( B \) and \( C \) satisfy the condition (ii). \( \square \)

The above three theorems tell us that knowing that a set is \( \leq_m^P \)-complete in \( \text{EXPTIME} \) does not give us any information about its approximability. The next theorem, which can be proved using a similar technique, shows that, for some \( \leq_m^P \)-complete sets in \( \text{EXPTIME} \), the situation may be even worse.

**Theorem 6.11.** There exists a \( \leq_m^P \)-complete set in \( \text{EXPTIME} \) such that for any \( \varepsilon > 0 \),

(i) \( (\forall B, C \in P) \left[ B \subseteq A, C \subseteq \overline{A} \Rightarrow \text{dens}_n(B \cup C) < \varepsilon \text{ i.o.} \right] \)

(ii) \( (\exists B, C \in P) \left[ B \subseteq A, C \subseteq \overline{A} \& \text{dens}_n(B \cup C) > 1 - \varepsilon \text{ i.o.} \right] \)

where i.o. is the abbreviation of "for infinitely many \( (n) \)."

**Proof.** Let \( A_1 \) be the set described in theorem 6.9. That is, \( A_1 \) is \( \leq_m^P \)-complete in \( \text{EXPTIME} \) and satisfies conditions (i) and (ii) in theorem 6.9.

Define two sequences of integers:

\[ a_n = 2^{(n^2-n)/2}, \text{ } n = 0, 1, \ldots \]
b_0 = 0, b_n = \sum_{i=0}^{n-1} (a_{2i+1} - a_{2i}), n = 1, 2, \ldots.

Then, the function f : N \rightarrow N defined by

f(n) = \text{the n-th element in the set } E = \bigcup_{n=0}^{\infty} \{x \mid a_{2n} \leq x < a_{2n+1}\},

is a polynomial time computable one-to-one correspondence between N and E. Moreover, f^{-1} is polynomial time computable.

Now, let A = f(A_1). A is the set we need.

First, it is easy to see that A is in \text{EXPTIME} by the following simple algorithm:

For given input x,

1. compute n and y \geq 0 such that x = a_n + y and x < a_{n+1}.

2. if n is odd, then reject x;

3. if n is even, then accept x if and only if f^{-1}(x) \in A_1.

Also, A_1 \leq^p_m A via f. So, A is \leq^p_m-complete in \text{EXPTIME}.

For condition (i), we check it in the following.

For each \varepsilon > 0, take n_0 so large that 2^{-(n_0+1)} < \varepsilon and

if B, C \in P, B \subseteq A_1, C \subseteq \overline{A}_1, then \text{dens}_m(B \cup C) < \varepsilon/2

for all m > n_0. Now, for any B, C \in P, B \subseteq A and C \subseteq \overline{A}, and any n, let

B_n = B \cap \{x \mid a_{2n} \leq x < a_{2n+1}\} \quad \text{and}

C_n = C \cap \{x \mid a_{2n} \leq x < a_{2n+1}\}.

Then

B \subseteq A \quad \text{and} \quad C \subseteq \overline{A}

\Rightarrow B_n \subseteq A \quad \text{and} \quad C_n \subseteq \overline{A}

\Rightarrow f^{-1}(B_n) \subseteq A_1 \quad \text{and} \quad f^{-1}(C_n) \subseteq \overline{A}_1
and 
\[ B, C \in \mathcal{P} \Rightarrow B_n, C_n \in \mathcal{P} \Rightarrow f^{-1}(B_n), f^{-1}(C_n) \in \mathcal{P}. \]

Thus, for \( n > n_0 \),
\[
\text{dens}_{a_{2n+1}} (B \cup C) \\
= \left| (B \cup C) \cap \{ x \mid x < a_{2n+1} \} \right| / a_{2n+1} \\
= (|B_n \cup C_n| + a_{2n}) / a_{2n+1} \\
\leq |B_n \cup C_n| / a_{2n+1} + 2^{-n} \\
= \left| f^{-1}(B_n) \cup f^{-1}(C_n) \right| / a_{2n+1} + 2^{-n} \\
= (|f^{-1}(B_n) \cup f^{-1}(C_n)| / b_{n+1}) * (b_{n+1} / a_{2n+1}) + 2^{-n} \\
\leq \text{dens}_{b_{n+1}} (f^{-1}(B_n) \cup f^{-1}(C_n)) + 2^{-n} < \varepsilon. \\
\]

Hence, for any \( B, C \in \mathcal{P} \), if \( B \leq A \) and \( C \leq \overline{A} \) then
\[ \text{dens}_n (B \cup C) < \varepsilon \text{ i.o.} \]

Condition (ii) is easily satisfied by taking \( B = \emptyset \) and
\[ C = \{ x \mid a_{2n-1} \leq x < a_{2n} \}. \]
Then \( \text{dens}_{a_{2n}} (B \cup C) > 1 - \varepsilon \) for all \( n \). Thus \( \text{dens}_n (B \cup C) > 1 - \varepsilon \text{ i.o.} \)

6.4. Summary and Open Questions

We have studied the polynomial time approximation to sets in \( \text{EXPTIME} \). In summary,

1. A set in \( \text{EXPTIME} \) may be so hard that its polynomial complexity core is exactly \( \Sigma^* \).

2. A \( \leq^p_{tt} \)-complete set in \( \text{EXPTIME} \) may be so hard that it is p-immune.
3. A $\leq^p_m$-complete set in EXPTIME cannot be p-immune.

4. The polynomial complexity core of a $\leq^p_m$-complete set in EXPTIME may be of any size.

5. The polynomial complexity core of a $\leq^p_m$-complete set in EXPTIME is larger than any p-sparse set.

What about the sets in NP? Can we show similar results?

Recently, P. Berman [32] and Meyer and Paterson [54] solved question 5 for the class of NP-complete sets. Namely, the following problems don't have p-sparse complexity cores unless $P = NP$:

- NP-complete problems,
- PSPACE-complete problems,
- the primality problems,
- the graph isomorphism problem, and
- the linear programming problem.

The other four questions remain open. We believe that any new discoveries on these questions will lead to better understanding of the structure of the intractable problems.
APPENDIX

Definitions of Optimization Problems

Binpacking: Given a finite list of rational numbers between 0 and 1 and a sequence of unit-capacity bins, find a packing of the numbers into the bins such that no bin contains a total exceeding 1 and the number of nonempty bins is minimized.

Exact cover: Given a finite collection of finite sets, find a sub-collection which covers the collection and has the least overlapping.

Feedback vertex: Given a directed finite graph, find the smallest subset of vertices such that every cycle of the graph contains a vertex in the subset.

Graph coloring: Given an undirected finite graph, find a way of coloring the vertices of the graph such that no two adjacent vertices have the same color and the number of colors used is minimized.

Knapsack: Given a positive integer M and a sequence Q of n pairs of positive integers \( \{(p_i, c_i)\} \), find a subsequence Q' of Q such that the sum of \( c_i \)'s in Q' does not exceed M and the sum of \( p_i \)'s in Q' is maximized.
Max clique: Given a graph, find the maximum subgraph which is a clique.

Max cut: Given an edge-weighted graph, find a subset of vertices such that the sum of weights of edges which is incident on exactly one vertex in the subset is maximized.

Max satisfiability: Given a set of disjunctive form clauses, all of whose disjuncts are literals or their negations, find a truth assignment to the literals which satisfies the most clauses.

Min clique cover: Given a graph, find a collection of subsets of vertices which forms a pairwise disjoint cover of the vertices and the number of cliques is minimized.

Min vertex cover: Given an undirected graph, find a minimum sized subsets of vertices such that every edge is incident on at least one vertex in the subset.

Primality: Given a positive integer, find out whether it is a prime number or not. (This is not an optimization problem and is not NP-complete if NP ≠ co-NP).

Set covering: Given a finite collection of finite set, find a subcollect of the sets which covers the collection and uses the fewest sets.
Set packing: Given a finite collection of finite sets, find a subcollection of pairwise disjoint sets such that the size of the subcollection is maximized.

Sum of subsets: Given a finite sequence of positive integers and a positive integer "goal", find a subsequence of integers whose sum is closest to, without exceeding, the goal.

Traveling salesman: Given a complete, edge-weighted graph, find a simple path which passes through every vertex and the total weight of the path is minimized.

2-partition: This is the special case of the sum of subsets problem with the goal equal to half of the total sum.
LIST OF REFERENCES

Part A


Part B


