# Stony Brook University

# ON THE COMPLEXITY OF REAL AND COMPLEX FUNCTIONS DEFINED ON THE TWO-DIMENSIONAL PLANE

A Dissertation Presented

by

**Fuxiang Yu**

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

**Doctor of Philosophy**

in

**Computer Science**

Stony Brook University

**August 2007**

# Stony Brook University

The Graduate School

Fuxiang Yu

We, the dissertation committee for the above candidate for the
Doctor of Philosophy degree, hereby recommend
acceptance of this dissertation.

**Ker-I Ko – Dissertation Advisor**

**Professor, Computer Science Department**

**Steven Skiena – Chairperson of Defense**

**Professor, Computer Science Department**

**Joseph S.B. Mitchell**

**Professor, Computer Science Department and Applied Mathematics Department**

**Xianfeng David Gu**
**Assistant Professor, Computer Science Department**

**Gary R. Mar – External Member**

**Associate Professor, Philosophy Department**

This dissertation is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

**On the Complexity of Real and Complex Functions Defined on the Two-dimensional Plane**

by

**Fuxiang Yu**

**Doctor of Philosophy**

in

**Computer Science**

Stony Brook University

2007

Computational complexity is a central area of theoretical computer science. Corresponding to the objects being studied, there are two categories: continuous complexity and discrete complexity. Since 1970s, many complexity theories have been developed on discrete computation, and a number of important results, especially those related to the $P$ versus $NP$ problem, have been obtained. On the other hand, although there is significant progress in the continuous side, it is fair to say that, compared with discrete complexity theory, more work could and should be done in continuous complexity theory. For instance, a fundamental issue in continuous complexity theory is the lack of a single, universal computational model that all researchers agree upon. Another major issue is that the computational complexity of many fundamental problems in computational mathematics is not known or not even attacked at all. An example is the computational complexity of analytic continuation problem.

The basic computational model we used in this dissertation is the oracle Turing machine model of Ko and Friedman, in which complexity is measured by the number of bit operations needed to output an approximation to the results within the required precision. In this dissertation we study the computational complexity of the following

fundamental problems in continuous mathematics.

(1) *Computing the analytic branches of the logarithm function and the square root function defined on a two-dimensional Jordan domain* (Ko and Yu [45]). The key result is as follow: When the boundary of the Jordan domain is polynomial-time computable, then the complexity of computing the analytic branches of the logarithm function is characterized by the complexity class $\#P$.

(2) *Operations on the NC and log-space functions* (Yu [82; 83]). We show that analyticity is critical for the two fundamental operations, integration and differentiation, to be closed in $NC$ or log-space, respectively. We also investigate the problem of finding all zeros of an analytic $NC$ or log-space function inside a Jordan curve. In addition, we discuss the expressive powers of different representations under $NC$ and Log-space.

(3) *Convex hulls of two-dimensional sets*(Ko and Yu [47]). We show that the convex hull of a $P$-computable Jordan domain $S$ is $NP$-recognizable but not necessarily polynomial-time recognizable if $P \neq NP$. We also show that the area of the convex hull of a $P$-computable Jordan domain $S$ is computable in polynomial time relative to an oracle function in $\#P$, and the area is a $\#P$ real number if $NP = UP$.

(4) *Circumscribed rectangles and squares*(Yu, Chou and Ko [85]). Our main results are: (a) The minimum-area circumscribed rectangles of a $P$-computable Jordan curve are not necessarily computable; (b) The complexity of the minimum area of all circumscribed rectangles of a $P$-computable Jordan curve is characterized by the complexity class $\Sigma_2^P$; and (c) If there exist a unique circumscribed square of a $P$-computable Jordan curve, it is computable but can be arbitrarily hard.

(5) *The pancake problem* (Yu [84]). We study the complexity of finding a straight

line that bisects simultaneously two bounded sets in the two-dimensional plane. We characterize its complexity by the complexity class $\#P$, under some thickness assumption and separability assumption on the sets in question. We have also obtained positive and negative results regarding some related problems such as the problem of bisecting one set in a given direction.

# Table of Contents

# List of Figures

## Acknowledgements

I would like to express my gratitude to all people who in some way have contributed to this thesis. Without them, this thesis will not be possible.

A special thank you goes to my advisor, Professor Ker-I Ko. I have known Professor Ko for many years and I have really learned many things from him. I have benefited much from his insights into computer science and mathematics. He taught me how to do research by recommending challenging problems, discussing and correcting my errors in detail and so on. He is always patient and active no matter whether we are discussing interesting problems or dealing with some details that seem boring in some sense. We published quite a few papers together. Besides being a nice advisor and teacher, Professor Ko is also like a good friend to me and helped me solve some nonacademic problems. It is really a great pleasure for me to get to know him and be his student.

I would also thank my coauthors and colleagues related to this research. These people include Professors Arthur Chou, Ning Zhong and many others. I had quite many interesting discussions with them and some nice ideas have been implemented by us together. I believe we can do better and better if we keep trying in the future.

In my early years in Stony Brook, I had been doing research with Professor Annie Liu on program analysis and transformation. From Professor Liu I learned a lot of techniques which I still use today. Under her supervision, we did some nice research which resulted in a few conference papers and technical reports. I always feel grateful to her and people in her group, including Professor Scott Stoller, and fellow students Tom, Nanjun and Katia.

My acknowledgments also go to the national science foundation and the department of computer science who have supported my research in these years. I would like to thank the secretaries Betty, Edwina, and Kathy.

I would like to thank my friends who have helped me with this thesis in some way. The names include, but are not limited to, Michael, Dongdong, Haodong, Dezhuang, Rui, Xuebo, Zhendong, and Ruoxi.

Last, but not the least, my thanks go to my family members, including my wife, Maggie, and my mother, sister, and parents in-law. My father, who was a farmer, a stone artist and a teacher with only one student, would be happy in heaven seeing that I have eventually reached here.

# Chapter 1

# Introduction

## 1.1 Motivation

This dissertation studies computational complexity of continuous problems. Although the main focus of computational complexity theory has been on discrete objects, most mathematical models in physics and engineering are based on the real number concept, and deal with continuous objects. Complexity theory of real functions is thus needed in order to understand the nature of computation about continuous data structures. In fact, Turing, in his classic paper [74], introduced the Turing machine to deal with integers as well as real numbers, and gave the first definition of computable real numbers. In the 1950s and 1960s, the basic notions in computable analysis, such as computable real functions and recursively measurable sets, had been established (see, for instance, Aberth [2],Lacombe [49], Goodstein [31], Grzegorczyk [33], and Ceitin [16]). More recently, the study of complexity of continuous problems has been developed more rapidly and gained much attention (see the monographs of Pour-El and Richards [63], Ko [40], Weihrauch [78], and Blum et al. [10], and the most recent expository paper of Braverman and Cook [12]).

This dissertation focuses on the computational complexity of real and complex

functions on the two-dimensional plane. We study the computational difficulty of some important fundamental continuous problems under the oracle Turing machine model of Ko and Friedman [42], which enables us to characterize the complexity of continuous problems with discrete complexity classes. The overall objective is to study the computational difficulty of these problems, so that we will gain insights into continuous complexity and its relation with discrete complexity theory.

## 1.2 Computational Models

There are many models for continuous computation, for instance, type-2 theory of effectivity (TTE) [78], the computational model of Blum, Shub and Smale (BSS) [10], information-based computational model [73], exact geometric computation(EGC) [80]. We will explain these models in details in Chapter 2. Here we briefly describe the oracle Turing machine model of Ko and Friedman [40; 42], which is the main model used in this dissertation.

The oracle Turing machine model of Ko and Friedman is based on the Turing machine and the complexity measure is the number of bit operations. The basic problem is how difficult, or equivalently how many computational resources (e.g., space and time) are required, to compute an approximation to a number with a required precision. To this end, we need to define the notion of feasibly computable real numbers and real functions. For example, we say a number $x \in [0, 1]$ is polynomial-time computable if there exist a polynomial function $p$ and a Turing machine $M$ such that for any input $n \in \mathbb{N}$, $M$ outputs an approximation $d$ to $x$ in time $p(n)$ such that $|x - d| < 2^{-n}$, and we identify the mathematical notion of polynomial-time computable real numbers with the intuitive notion of feasibly computable real numbers.

The notion of feasibly computable real functions is more complicated. We use the notion of polynomial-time oracle Turing machines to capture this concept. A real

function $f : [0,1] \to \mathbb{R}$ is computable if its values at any point can be approximated by an oracle Turing machine $M$. In order to approximate $f(x)$ with a precision $2^{-n}$ for some $x \in [0,1]$, $M$ takes $n$ as the input and asks an oracle $\phi$ to provide the information about $x$. The oracle $\phi$ can provide approximations to $x$ with any precision, but different precisions require different amount of computational resources. In general, in order to compute a more accurate approximation to $f(x)$, a more accurate approximation to $x$ is needed. The complexity will be measured in terms of precision $n$ according to the resources used. We delay the technical details till Section 2.4.3.

This model preserves theoretical properties of real numbers and real functions (e.g., the set of computable real numbers forms a real closed field). It also reflects the practice in numerical analysis (e.g., it is undecidable in this theory to determine whether two given real numbers are equal). In addition, it connects discrete complexity theory and continuous computation in a natural way, and allows us to use notions of discrete complexity theory to study the inherent complexity of continuous problems.

## 1.3 Main Topics and Contributions

We focus on a few fundamental mathematical problems in the areas of geometry, analysis and topology. These problems mainly concern with functions defined on the two-dimensional plane and are often much more complicated than the one-dimensional case. We have obtained tight upper and lower complexity bounds, if not the exact complexity, of most problems. To obtain a tight upper bound, a suitable algorithm is needed, which involves with the techniques from both algorithm design and numerical analysis. To obtain a tight lower bound, a suitable construction of reductions from a discrete problem of known complexity is needed.

In addition, we point out that computability of these problems or related problems have also been studied and some results are interesting.

**The logarithm and the square root functions [45].** Analytic continuation is an important operation in computational complex analysis. We focus on a special case, the problem of computing single-valued, analytic branches of the logarithm and square root functions defined on a two-dimensional simply connected domain $S$. This problem is trivial when we consider the complexity of logarithm and square root functions defined on the one-dimensional real line. However, for the two-dimensional case, there is no simple algorithm to achieve this task.

We prove that, if the boundary $\partial S$ of $S$ is a polynomial-time computable Jordan curve, the complexity of these problems can be characterized by counting classes $\#P$, $MP$ (or $MidBitP$), and $\oplus P$: The logarithm problem is polynomial-time solvable if and only if $FP = \#P$. For the square root problem, it is shown to have the upper bound $P^{MP}$ and lower bound $P^{\oplus P}$. That is, if $P = MP$ then the square root problem is polynomial-time solvable, and if $P \neq \oplus P$ then the square root problem is not polynomial-time solvable.

***NC* and log-space functions defined on $\mathbb{R}$ or $\mathbb{C}$ [82; 83].** Recall that $NC$ is the class of problems that can be solved efficiently by parallel computers. If a function $f$ defined on $\mathbb{R}$ or $\mathbb{C}$ is $NC$ computable, what is the complexity of computing the derivative $f'$ (if applicable) and the integral $\int f$? What about the problem of finding zeros of such functions? These are fundamental problems regarding continuous parallel time complexity. Our results demonstrate that the property of analyticity of the function $f$ is critical to keep the complexity of these operators within $NC$.

We show that for $NC$ real functions, differentiation and integration are infeasible, but analyticity helps to reduce the complexity. For example, the integration of a log-

space computable real function $f$ is as hard as $\#P$, but if $f$ is an analytic function, then the integration is log-space computable. As an application, we study the problem of finding all zeros of an $NC$ analytic function inside a Jordan curve and show that, under a uniformity condition on the function values of the Jordan curve, the zeros are all $NC$ computable.

A related question is representations of real numbers in parallel complexity theory. We show that the relations among the representations are more complicated than those in sequential complexity theory. For example, the Cauchy function representation and the general left cut representation are equivalent with respect to polynomial-time computability, but not equivalent with respect to $NC$ computability, unless $P = NC$.

**Convex hulls [47].** There are many nice algorithms to compute the convex hull of a set $S$ of finite points. In this dissertation we study this problem in the polynomial-time complexity theory of real functions based on the oracle Turing machine model.

We show that the convex hull of a two-dimensional Jordan domain $S$ is not necessarily recursively recognizable even if $S$ is polynomial-time recognizable. On the other hand, if the boundary of a Jordan domain $S$ is polynomial-time computable, then the convex hull of $S$ must be $NP$-recognizable, and it is not necessarily polynomial-time recognizable if $P \neq NP$. We also show that the area of the convex hull of a Jordan domain $S$ with a polynomial-time computable boundary can be computed in polynomial time relative to an oracle function in $\#P$. On the other hand, whether the area itself is a $\#P$ real number depends on the open question of whether $NP = UP$.

**Circumscribed rectangles and squares [85].** We say a rectangle (or a square) $R$ circumscribes a Jordan curve $\Gamma$ if (1) every point of $\Gamma$ is inside or on $R$; and (2) every side of $R$ intersects $\Gamma$. Assume that $\Gamma$ is polynomial-time computable.

We are concerned with the problem of finding the minimum area of circumscribed rectangles at all angles of $\Gamma$, and the similar problem for circumscribed squares.

We show that a bounded domain with a polynomial-time computable Jordan curve $\Gamma$ as the boundary may not have a computable minimum-area circumscribed rectangle. We also show that the problem of finding the minimum area of a circumscribed rectangle of a polynomial-time computable Jordan curve $\Gamma$ is equivalent to a discrete $\Sigma_2^P$-complete problem. For the related problem of finding the circumscribed squares of a Jordan curve $\Gamma$, we show that for any polynomial-time computable Jordan curve $\Gamma$, there must exist at least one computable circumscribed square (not necessarily of the minimum area), but this square may have arbitrarily high complexity.

**The pancake problem [84].** The pancake theorem, in the classic form, states that the areas of two plane sets $S_1$ and $S_2$ can be bisected simultaneously by a line $L$. This theorem is fundamental in topology in the sense that it is equivalent to the Brouwer fixed point theorem. We are interested in the complexity of finding the bisecting line $L$. We also study the basic problem of bisecting a set at a given direction.

Our main results are: (1) the complexity of bisecting a nice (thick) polynomial-time approximable set at a given direction can be characterized by the counting class $\#P$; (2) the complexity of bisecting simultaneously two linearly separable nice (thick) polynomial-time approximable sets can also be characterized by the counting class $\#P$; and (3) for either of these two problems, without the thickness condition and the linear separability condition (for the two-set case), it is arbitrarily hard to compute the bisector, even if it is unique.

# Chapter 2

# Computational Complexity of Real-valued Problems

In this chapter, we survey the main methods and models of computational complexity of real-valued problems. We explain in detail the oracle Turing machine model of Ko and Friedman, since this is the model we will use. Basic descriptions of the discrete complexity theory and computability theory are also included.

## 2.1 Notations

This dissertation involves notions used in both discrete computation and continuous computation. The basic computational objects in discrete computation are integers and strings in $\{0,1\}^*$. A *language* is a subset of $\{0,1\}^*$; we also use *sets* interchangeably with *languages* when no confusion is caused. The length of a string $w$ is denoted $\ell(w)$. We write $\langle w_1, w_2\rangle$ to denote a fixed pairing function on $w_1$ and $w_2$; more precisely, $\langle w_1, w_2\rangle = 0^{\ell(w_1)}1w_1w_2$ for any $w_1, w_2 \in \{0,1\}^*$. We write $||S||$ to denote the number of elements in a (finite) set $S$, and $S^c$ to denote the complement of $S$ relative to $\{0,1\}^*$ (i.e., $S^c = \{0,1\}^* - S$).

The basic computational objects in continuous computation are *dyadic rationals* $\mathbb{D} = \{m/2^n : m \in \mathbb{Z}, n \in \mathbb{N}\}$, and we denote $\mathbb{D}_n = \{m/2^n : m \in \mathbb{Z}\}$. Each dyadic rational $d$ has infinitely many binary representations with arbitrarily many trailing zeros. For each such representation $s$, we write $\ell(s)$ to denote its length. If the specific representation of a dyadic rational $d$ is understood (often the shortest binary representation), then we write $\ell(d)$ to denote the length of this representation.

We use $\mathbb{R}$ to denote the real line (and the class of real numbers) and $\mathbb{R}^2$ the 2-dimensional plane. We will use the complex plane $\mathbb{C}$ instead of $\mathbb{R}^2$ when dealing with analytic functions. We often use letters in boldface font, such as $\mathbf{z}$, or two letters in brackets, such as $\langle x, y\rangle$, to represent a point in $\mathbb{R}^2$. For any point $\mathbf{z} \in \mathbb{R}^2$ and any set $S \subseteq \mathbb{R}^2$, we let $\mathrm{dist}(\mathbf{z}, \mathrm{S})$ be the distance between $\mathbf{z}$ and $S$; that is, $\mathrm{dist}(\mathbf{z}, \mathrm{S}) = \inf\{|\mathbf{z} - \mathbf{z}'| : \mathbf{z}' \in \mathrm{S}\}$, where $|\cdot|$ denotes the absolute value. The complement of a set $S \subseteq \mathbb{R}^2$ relative to $\mathbb{R}^2$ is written as $S^c$ (i.e., here $S^c = \mathbb{R}^2 - S$), the closure of a set $S$ is written as $\overline{S}$, and the boundary of a set $S$ is written as $\partial S$.

## 2.2 Computability, Turing Machines and Oracle Turing Machines

Computability theory studies what computers can and cannot do. Introduced by Allan Turing [75], Turing machines are one of the main abstractions used in computability theory. A Turing machine, in a few words, is a finite-state machine with an external storage medium such as a tape (see, e.g., Hopcroft and Ullman [38] and Du and Ko [26]). According to the Church-Turing Thesis, Turing machines are equivalent to all ordinary computers in terms of theoretical computational power.

Because Turing machines have the ability to *back up*(i.e., shift to left) in their input tape, it is possible for a Turing machine to run forever on some inputs. We say

that a Turing machine can decide a language if it eventually will halt on all inputs and give an answer. A language that can be so decided is called a *recursive language*. [1] We can further describe Turing machines that will eventually halt and give an answer for any input in a language, but which may run forever for input strings which are not in the language. Such Turing machines could tell us that a given string is in the language, but we may never be sure based on its behavior that a given string is not in a language, since it may run forever in such a case. A language which is accepted by such a Turing machine is called an *r.e.* (short for *recursively enumerable*) language. There exists an r.e. language that is not recursive.

There are both deterministic Turing machines and nondeterministic Turing machines (see, e.g., Du and Ko [26]). The transition function of a nondeterministic Turing machine may take multiple values, which means for an input to the machine, there may be multiple computation paths. As nondeterministic Turing machines can be simulated by deterministic Turing machines, there is no difference between them regarding computability. However, as far as complexity is concerned, they may be quite different.

An oracle Turing machine is an ordinary Turing machine enhanced with a state, or a black box, called an *oracle*. The computation of an oracle Turing machine is just like an ordinary Turing machine, except that at the oracle state, an oracle is queried to give an answer to proceed. Oracle Turing machines are more powerful than Turing machines if the oracles are undecidable problems such as the halting problem. In the theory of computational complexity, often problems of certain complexity are used as oracles.

A *deterministic* (*function-*)*oracle Turing machine* is a deterministic Turing machine equipped with an additional *query* tape and two additional states: the *query* state and the *answer* state. The oracle Turing machine $M$ works as follows: First,

[1] In the literature, *recursive* is used interchangeably with *computable*.

on input $x$ with oracle function $f$ (which is a total function on $\{0,1\}^*$), it begins the computation at the initial state and behaves exactly like the ordinary Turing machine when it is not in any of the special state. The machine is allowed to enter the query state to make queries to the oracle $f$, but it is not allowed to enter the answer state from any ordinary state. Before it enters the query state, machine $M$ needs to prepare the query string $y$ by writing the string $y$ on the query tape. After the machine enters the query state, the computation is taken over by the oracle $f$, which will (1) read the string $y$ on the query tape; (2) replace $y$ with string $f(y)$; and (3) put the machine into the answer state. Then the machine continues from the answer state as usual. The actions taken by the oracle count as only one unit of time. We write $M^f(x)$ to denote the computation of $M$ on input $x$ using oracle $f$.

A *nondeterministic* (*function-*)*oracle nondeterministic Turing machine* is a nondeterministic Turing machine equipped with an additional query tape and two additional states: the query state and the answer state. The computation of a nondeterministic oracle Turing machine is similar to that of a deterministic oracle Turing machine, except that at each nonquery state a nondeterministic oracle Turing machine can make a nondeterministic move.

A set-oracle Turing machine is a special kind of function-oracle Turing machine with the oracle being a set (thus the answer to a query to the oracle is always "yes" or "no"), since a set can be viewed as a function from $\{0,1\}^*$ to $\{0,1\}$.

## 2.3 Discrete Complexity Theory

Computational complexity theory studies the resources, or cost, of computation required to solve a given computational problem. Discrete complexity theory studies the complexity of discrete computational problems. There are various computational models in discrete complexity theory, and we mainly use the Turing machine model

and the circuit model. Here we present the theories based on these two models; for more details, see any standard computational complexity textbook, such as Du and Ko [26] and Papadimitriou [59].

### 2.3.1 $NP$ theory

The Turing machine-based complexity theory plays an important role in computational complexity theory. Based on the Turing machine model, a number of complexity classes, such as $P$, $NP$, $\#P$, $PSPACE$ and $LOGSPACE$, have been defined, and their relationship has been widely studied. Among all these relations, the $P$ versus $NP$ question, i.e., the question of whether $P$ equals $NP$, is the central issue of the complexity theory.

We first define complexity classes of languages. The two fundamental complexity classes are the complexity classes $P$ and $NP$ defined as follows.

**$P$:** the class of sets accepted by deterministic polynomial-time Turing machines.

**$NP$:** the class of sets accepted by nondeterministic polynomial-time Turing machines.

Then we use oracle Turing machines to define the *polynomial-time hierarchy*.

Let $A$ be a set and $\mathcal{C}$ be a complexity class. We let $NP^A$ denote the class of sets accepted by polynomial-time nondeterministic oracle Turing machines using the oracle $A$, and let $NP^{\mathcal{C}}$ denote the class of sets accepted by polynomial-time nondeterministic oracle Turing machines using an oracle $B \in \mathcal{C}$ (i.e., $NP^{\mathcal{C}} = \bigcup_{B \in \mathcal{C}} NP^B$). For a complexity class $\mathcal{C}$, we use *co-*$\mathcal{C}$ to denote the complexity class of sets whose complements are in $\mathcal{C}$, i.e., *co-*$\mathcal{C} = \{A \subseteq \{0,1\}^* : A^c \in \mathcal{C}\}$.

**Definition 2.3.1** *For integers $n \in \mathbb{N}$, complexity classes $\Delta_n^P$, $\Sigma_n^P$, and $\Pi_n^P$ are defined*

*as follows:*

$$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P,$$
$$\Sigma_{n+1}^P = NP^{\Sigma_n^P},$$
$$\Pi_{n+1}^P = co\text{-}\Sigma_{n+1}^P,$$
$$\Delta_{n+1}^P = P^{\Sigma_n^P}, \quad n \geq 0.$$

*The class $PH$ is defined as the union of $\Sigma_n^P$ over all $n \geq 0$.*

Thus, $\Sigma_1^P = NP$, $\Sigma_2^P = NP^{NP}$, and $\Sigma_3^P = NP^{NP^{NP}}$, and so on.

The above are complexity classes defined based on running time. There are also complexity classes defined based on space usage.

**$PSPACE$:** (or, $NPSPACE$) is the complexity class of sets accepted by polynomial-space deterministic (or respectively, nondeterministic) Turing machines.

**$LOGSPACE$:** (or, $NLOGSPACE$) is the complexity class of sets accepted by log-space deterministic (or respectively, nondeterministic) Turing machines.

It is well known that $PSPACE = NPSPACE$. We often write $L$ for $LOGSPACE$ and $NL$ for $NLOGSPACE$.

Accompanying each complexity class $\mathcal{C}$ of languages, there is a corresponding complexity class $F\mathcal{C}$ of functions (mapping strings to strings), which uses the same amount of resources but the outputs are strings instead of "yes" or "no". For example, $FP$ is the class of functions computable by deterministic polynomial-time Turing machines.

Next we define the counting class $\#P$ and some classes related to it (sometimes we call all these classes *counting classes* to emphasize the relationship between them). These classes occur often in complexity theory of real analysis, because the complexity of integration is closely related to $\#P$ (see Ko [40]).

**$\#P$:** the class of functions that count the number of accepting paths of nondeterministic polynomial-time machines.

$\#L$**:** the class of functions that count the number of accepting paths of nondeterministic log-space machines.

$\#NP$**:** (or, $\# \cdot NP$) [2] the class of functions $\phi : \{0,1\}^* \to \mathbb{N}$ with the following property: There exist a set $B \in NP$ and a polynomial function $p$ such that, for any $w \in \{0,1\}^*$,

$$\phi(w) = \|\{u \in \{0,1\}^* : \ell(u) = p(\ell(w)), \langle w, u\rangle \in B\}\|.$$

$\oplus P$**:** the class of sets $A$ for which there exists a nondeterministic polynomial-time Turing machine $M$ such that for all $x$, $x \in A$ if and only if there are an odd number of accepting paths for $x$ in $M$; equivalently, a set $A$ is in $\oplus P$ if there exists a function $G \in \#P$ such that for all $x$, $x \in A$ if and only if $G(x) = 1 \pmod 2$.

$MP_b$**:** the class of sets $A$ for which there exist a function $G \in \#P$ and a function $\phi \in FP$ such that for all $x$, $x \in A$ if and only if the $\phi(x)$-th bit in the $b$-ary representation of $G(x)$ is not zero, where $b$ is an integer greater than one.

$MP$**:** the union of $MP_b$ over all $b \geq 2$.

We also use the following complexity class $UP$.

$UP$**:** the class of sets that are accepted by nondeterministic polynomial-time Turing machines that have, on any input, at most one accepting computation.

[2] In the original paper of Valiant [76], the notation $\#NP$ was defined to mean the class $\#P^{NP}$. Hemaspaandra and Vollmer [34] pointed out that, in view of the characterization of $\#P$ of Theorem 2.3.2(c) below, it appears to be more appropriate to define $\#NP$ to mean the class we defined here, and proposed, in a general framework, the notation $\# \cdot NP$ for this class. Here, we use $\#NP$ for its simplicity.

The relations between the complexity classes defined above have been studied since 1970s, which we summarize below; for detailed relations and proofs, see for example, the textbook by Du and Ko [26]. Note that whether any of the inclusions $\subseteq$ below, including $P \subseteq NP$, is proper, is a major question in the $NP$ theory. It can be seen that $\#P$ is a very powerful class.

**Theorem 2.3.2** *(a) A set $A \subseteq \{0,1\}^*$ is in NP if and only if there exist a set $B \in P$ and a polynomial function $p$ such that, for any $w \in \{0,1\}^*$,*

$$w \in A \iff (\exists u, \ell(u) = p(\ell(w)))\, \langle w, u\rangle \in B.$$

*(b) A set $A \subseteq \{0,1\}^*$ is in UP if and only if there exist a set $B \in P$ and a polynomial function $p$ such that, for any $w \in \{0,1\}^*$,*

$$\begin{aligned} w \in A &\iff (\exists u, \ell(u) = p(\ell(w)))\, \langle w, u\rangle \in B \\ &\iff (\exists \text{ a unique } u, \ell(u) = p(\ell(w)))\, \langle w, u\rangle \in B. \end{aligned}$$

*(c) A function $\phi : \{0,1\}^* \to \mathbb{N}$ is in $\#P$ if and only if there exist a set $B \in P$ and a polynomial function $p$ such that, for any $w \in \{0,1\}^*$,*

$$\phi(w) = \|\{u \in \{0,1\}^* : \ell(u) = p(\ell(w)), \langle w, u\rangle \in B\}\|.$$

*(d) For all $k > 0$, $\Sigma_k^P \cup \Pi_k^P \subseteq \Delta_{k+1}^P \subseteq \Sigma_{k+1} \cap \Pi_{k+1}^P \subseteq PH \subseteq PSPACE$.*

*(e) $L \subseteq P \subseteq PSPACE$, $L \neq PSPACE$.*

*(f) $PH \subseteq P^{\#P}$.*

*(g) $P \subseteq UP \subseteq NP \subseteq P^{\#P}$.*

*(h) $P = UP$ iff $\#P = \#NP$.*

A set $T \subseteq \{0,1\}^*$ is called a *tally set* if $T \subseteq \{0\}^*$. For a complexity class $\mathcal{C}$ of sets of strings in $\{0,1\}^*$, we let $\mathcal{C}_1$ denote the complexity class $\{A \in \mathcal{C} : A \subseteq \{0\}^*\}$. Similarly, for a complexity class $\mathcal{FC}$ of functions from $\{0,1\}^*$ to $\{0,1\}^*$, we let $\mathcal{FC}_1$ denote the complexity class $\{F \in \mathcal{FC} : \text{the domain of } F \text{ is } \{0\}^*\}$. In this dissertation, we will use complexity classes $FP_1$, $P_1$, $\#P_1$, $NC_1$, $L_1$ and so on.

### 2.3.2 Circuit complexity

Circuit complexity is another complexity theory that we will study in this dissertation. While *NP* theory focuses on feasible sequential complexity, circuit complexity focuses on feasible parallel-time complexity.

The model for circuit complexity is the Boolean circuit model. A Boolean circuit is a directed acyclic graph with leaves being input gates and roots being output gates; each non-leaf vertex in a Boolean circuit is a Boolean function AND, OR or NOT. For each $n, m \in \mathbb{N}$, a Boolean circuit $C$ with $n$ input gates and $m$ output gates computes a function from $\{0,1\}^n$ to $\{0,1\}^m$.

For $i \geq 0$, we let $NC^i$ be the class of languages $A \subseteq \{0,1\}^*$ for which there exists a family $\{C_n\}$ of Boolean circuits with the following properties (see, e.g., Du and Ko [26]):

(a) There exists a Turing machine $M$ that constructs (the encoding of) each $C_n$ in space $O(\log n)$.

(b) For all $n > 0$, $C_n$ has $n$ input nodes and accepts $A_n = A \cap \{0,1\}^n$.

(c) There exist a polynomial function $p$ and a constant $k > 0$, such that for all $n$, $size(C_n) \leq p(n)$ and $depth(C_n) \leq k \log^i n$.

We call $\{C_n\}$ an $NC^i$ *circuit family*. We let $NC$ be the union of $NC^i$ for all $i \geq 0$.

Condition (a) above assures that the circuit family $\{C_n\}$ is log-space uniform. This condition is very strong. While integer addition and multiplication are easily seen to be in $NC^1$ and $L$, integer division and iterated integer multiplication are much harder, and were proved to be in $NC^1$ and $L$ only a few years ago (see Cook et al. [7] and Chiu et al. [17]).

We summarize below the relations between complexity classes $L$, $NC$, $P$ and so on. The interested readers are referred to Du and Ko [26] and Papadimitriou [59] for

general properties of these complexity classes and to Alvarez and Jenner [4] for that of $\#L$.

**Theorem 2.3.3**

$$NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq NC \subseteq P \subseteq NP$$

*and*

$$\#L \subseteq FNC^2 \subseteq FP \subseteq \#P.$$

Whether any of the above inclusions is proper is unknown. Note that $P$ is considered the class of feasible sequentially solvable questions, and $NC$ is considered the class of feasible parallelizable questions. Whether $NC$ equals $P$ is another important question besides the $P$ versus $NP$ question in complexity theory.

## 2.4 Complexity of Real Numbers and Functions

From different viewpoints, researchers have proposed many models for continuous computation. We will use the oracle Turing machine model of Ko and Friedman, because it connects it with the solid Turing machine-based discrete complexity theory (i.e., $NP$ theory), which makes it easy to gain insights to the world of continuous computation. In this and next sections, we will explain continuous complexity theory based on the oracle Turing machine model. Other models, including models also based on the (discrete) Turing machine model and models based on other computational devices, will be discussed in the last section of this chapter.

### 2.4.1 The Oracle Turing Machine Model of Ko and Friedman

The oracle Turing machine model of Ko and Friedman [40; 42] uses the Turing machine as the basic model of computation. In this model, the complexity of a computable real

number $x$ is measured as the number of bit operations to find a binary approximation $d$ to $x$ with a required precision. This extends Turing's notion of computable real numbers that a real number $x$ is computable if there exists a Turing machine that can compute an approximation $d$ to $x$ with any required precision.

We also describe Hoover's Boolean circuit model [36] for $NC$ real functions, which is consistent with the model of Ko and Friedman.

**Representations of real numbers.** A real number has a few basic representations. The most basic one is the *Cauchy function representation*. We say a function $\phi : \mathbb{N} \to \mathbb{D}$ *binary converges to* a real number $x$, or is *a Cauchy function representation of* (or, simply, *represents*) $x$, if (i) for all $n \geq 0$, $\phi(n) \in \mathbb{D}_n$, and (ii) for all $n \geq 0$, $|x - \phi(n)| \leq 2^{-n}$. Among the set $CF_x$ of all Cauchy function representations of $x$, there is a unique function $b_x : \mathbb{N} \to \mathbb{D}$ that binary converges to $x$ and satisfies the condition $x - 2^{-n} < b_x(n) \leq x$ for all $n \geq 0$. We call this function $b_x$ the *standard Cauchy function* of $x$. We say two functions $\phi_x, \phi_y : \mathbb{N} \to \mathbb{D}$ binary converge to (or represent) a point $\mathbf{z} := \langle x, y \rangle \in \mathbb{R}^2$ if $\phi_x$ and $\phi_y$ binary converge to $x$ and $y$, respectively.

For each Cauchy function representation $\phi : \mathbb{N} \to \mathbb{D}$ of a real number $x$, there is an associated (*general*) *left cut representation*, namely, the set $L_\phi = \{d \in \mathbb{D}_n : d \leq \phi(n), n \geq 1\}$. Note that the set $L_\phi$ also uniquely determines the Cauchy function $\phi$ since $\phi(n) = \max\{\mathbb{D}_n \cap L_\phi\}$ for all $n \geq 1$. In other words, for a real number $x$, there is an 1-1 correspondence between the set of its Cauchy function representations and the set of its left cuts. Meanwhile, for a left cut $L$ of a real number $x$, there exists exactly one Cauchy function representation $\phi$ of $x$ such that $L_\phi = L$; more precisely, the function $\phi$ satisfies $\phi(n) = \max(\mathbb{D}_n \cap L)$. The (general) left cut of $x$ that is associated with the standard Cauchy function $b_x$ of $x$ is called the *standard left cut* of $x$.

### 2.4.2 Computable Real Numbers

The computability and complexity of a real number can be defined according to the computability and complexity of its Cauchy function and left cut representations. It may be natural to use the Cauchy function representation since a Cauchy function representation of a real number writes out its approximations directly; however, the left cut representation is more useful when we consider complexity classes $\mathcal{C}$ of sets, such as $NP$ and $\Sigma_2^P$, instead of complexity classes of functions.

**Definition 2.4.1** (1) *A real number $x$ is said to be* computable *if it has a computable left cut.*

(2) *A real number $x$ is said to be a* left $\mathcal{C}$-real number *if it has a left cut in $\mathcal{C}$, and $x$ is a* right $\mathcal{C}$-real number *if it has a* right cut (*the complement of a left cut*) *in $\mathcal{C}$, where $\mathcal{C}$ is a complexity class of sets.*

Note that a real number $x$ is computable iff it has a computable Cauchy function representation. The prefixes *left* and *right* in the above definition are necessary since, for many complexity classes $\mathcal{C}$ (e.g., $\mathcal{C} = NP$), it is still not known whether the complement *co-*$\mathcal{C}$ of $\mathcal{C}$ is identical to $\mathcal{C}$ or not. As $P =$ *co-*$P$, we can say a number $x$ is a $P$-real number, or equivalently, $x$ is *polynomial-time computable*, if $x$ is a left or right-$P$ real number. We remark that a real number $x$ is polynomial-time computable iff it has a polynomial-time computable Cauchy function representation since, from a polynomial-time computable left cut $L$ of $x$, we can compute the Cauchy function $\phi$ of $x$ with which $L$ is associated in polynomial time using binary search (recall that $\phi(n) = \max(\mathbb{D}_n \cap L)$ for all $n \geq 1$).

There are non-computable real numbers. This is because, there are uncountably many numbers on the real line, but there are only countably many Turing machines and thus only countably many numbers can be computed. On the other hand, if one

can describe a number, it often turns out that the number is computable, since the description often implies an algorithm.

We give below a few examples to demonstrate the concepts.

**Example 2.4.2** *Show that $\pi$ is computable, polynomial-time computable and log-space computable.*

*proof.* There are a great number of mathematical methods to compute $\pi$, and all of them can be converted into a proof that $\pi$ is computable in Definition 2.4.1. We only need to show that $\pi$ is log-space computable in order to show $\pi$ is computable and polynomial-time computable, since $L \subseteq P$.

We use the Bailey-Borwein-Plouffe formula [6]:

$$\pi = \sum_{k=0}^{\infty} 16^{-k}(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}).$$

A log-space algorithm to compute $\pi$ with a required precision is as follows.

**Input:** a positive integer $n$, which means that the required precision is $2^{-n}$.

**Approximation.** We use $\sum_{k=0}^{n} 16^{-k}(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6})$ to approximate $\pi$, which introduces an error $\leq 16^{-(n+1)}$. For each $k \in \{0, 1, \cdots, n\}$, a dyadic approximation $a_k$ to the term $16^{-k}(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6})$ $(k \in \{0, 1, \cdots, n\})$ with error $\leq 16^{-n}$ can be computed in $O(\log n)$ space, since integer addition, multiplication and division are all log-space computable (see Cook et al. [7] and Chiu et al. [17]). It takes $O(\log n)$ space to find the sum $\sum_{k=0}^{n} a_k$. □

Many other "$\pi$ formulas" lead to log-space algorithms. A further analysis shows that it takes linear time to compute the $n$-th bit of $\pi$. However, whether there is a linear algorithm to compute the first $n$ bits of $\pi$ (in other words, whether $\pi$ is computable in *real time*), remains an open question.

Many other well-known mathematical constants are also polynomial-time computable or even log-space computable. It is easy to show that the natural logarithm base $e = exp(1)$ is also log-space and polynomial-time computable. The Euler-Mascheroni constant $\gamma$, defined by $\gamma := \lim_{n\to\infty}(\sum_{k=1}^{n} \frac{1}{k} - \ln(n))$, is also log-space and polynomial-time computable; however, the proof is nontrivial (see, e.g., Gourdon et al. [32] for a survey of this problem).

**Example 2.4.3** *Show that there is a real number that is computable in $2^{O(n)}$ time but is not polynomial-time computable.*

*Proof.* Let $A \in \{0\}^*$ be a language that is computable in $2^{O(n)}$ time but is not polynomial-time computable (for the existence of such a language, see, e.g., Hopcroft and Ullman [38]). Note that $A$ must contain infinitely many strings. Let $\chi_A$ be the characteristic function of $A$. Define

$$x = \sum_{k=0}^{\infty} \chi_A(0^k)2^{-2k}.$$

The number $x$ is computable in $2^{O(n)}$ time, because for any $n \in \mathbb{N}$, we can let the recognizer of $A$ check in $2^{O(n)}$ time whether $0^k$'s $(0 \leq k \leq n/2)$ are in $A$ (i.e., to compute $\chi_A(0^k)$), and output $d = \sum_{0\leq k\leq n/2} \chi_A(0^k)2^{-2k}$ as an approximation to $x$ with error $\leq 2^{-n}$. Next we prove that $x$ is not polynomial-time computable.

Assume, by way of contradiction, that $x$ is polynomial-time computable by a machine $M$. To check whether $0^n \in A$, let $M$ compute an approximation $d \in \mathbb{D}_{2n+1}$ to $x$ such that $|d - x| \leq 2^{-(2n+1)}$. Let $d' = \sum_{0\leq k\leq n} \chi_A(0^k)2^{-2k}$. We have $d' \leq x < d' + 2^{-(2n+1)}$ and $d \geq d'$ (because $d'$ is the minimum possible approximation in $\mathbb{D}_{2n+1}$ to $x$ with the precision $2^{-(2n+1)}$). More precisely, $d' \leq d \leq d' + 2^{-(2n+1)}$. Note that the $(2n+1)$-st bit of $d'$ is 0 and thus is no more than the $(2n+1)$-st bit of $d$. If the $2n$-th bit of $d$ is different from that of $d'$, we must have $d - d' \geq 2^{-2n} > 2^{-(2n+1)}$, which is a contradiction to the fact that $d \leq d' + 2^{-(2n+1)}$. Therefore, the $2n$-th bit of

$d$ is the same as that of $d'$, which means that $\chi_A(0^n)$ can be extracted from $d$. This proves that $A$ is polynomial-time computable, which is a contradiction. □

Suppose $\mathcal{C}$ is a complexity class of sets, such as $P$, $NP$, and $L$. We use $\mathcal{C}_{\mathbb{R}}$ to denote the class of all $\mathcal{C}$ computable real numbers. We remark that the hierarchy of complexity of real numbers is closely related to the hierarchy of discrete complexity classes. For example, Ko [40, Theorem 3.10] proved that $P = NP \Rightarrow P_{\mathbb{R}} = NP_{\mathbb{R}} \Rightarrow P_1 = NP_1$. The exact classification of $NP$ real numbers is left open. We would like to find two natural discrete complexity classes such that $P_{\mathbb{R}} = NP_{\mathbb{R}}$ iff these two complexity classes coincide.

### 2.4.3 Computable Real Functions

To compute a real-valued function $f : [0, 1] \to \mathbb{R}$, we use an oracle Turing machine as the computational model. Namely, the function $f$ is computable if there exists a Turing machine $M$ such that for any oracle function $\phi$ that binary converges to a number $x \in [0, 1]$ and any integer $n \in \mathbb{N}$, $M$ outputs a dyadic number $e$ such that $|e - f(x)| < 2^{-n}$; furthermore, we say $f$ is polynomial-time computable if the machine $M$ that computes $f$ is a polynomial-time machine. Here we still use $M^{\phi}(n)$ to denote the computation of $M$ on input $n$ with an oracle $\phi$; if $\phi$ represents a dyadic number $d$, we also write $M^d(n)$ for $M^{\phi}(n)$.

Figure 2.1 illustrates the computation of $f(x)$. To compute an approximation to $f(x)$, the machine $M$ queries the oracle to obtain an approximation $d$ to $x$, then $M$ computes an approximation $e$ to $f(d)$, and outputs $e$ as an approximation to $f(x)$. Since $|f(x) - e| \leq |f(x) - f(d)| + |f(d) - e|$, if the errors $|f(x) - f(d)|$ and $|f(d) - e|$ are well controlled, so is $|f(x) - e|$.

We give an equivalent definition for this concept below that does not require the use of oracle Turing machines. Note that if a function $f$ is computable, then $f$ has

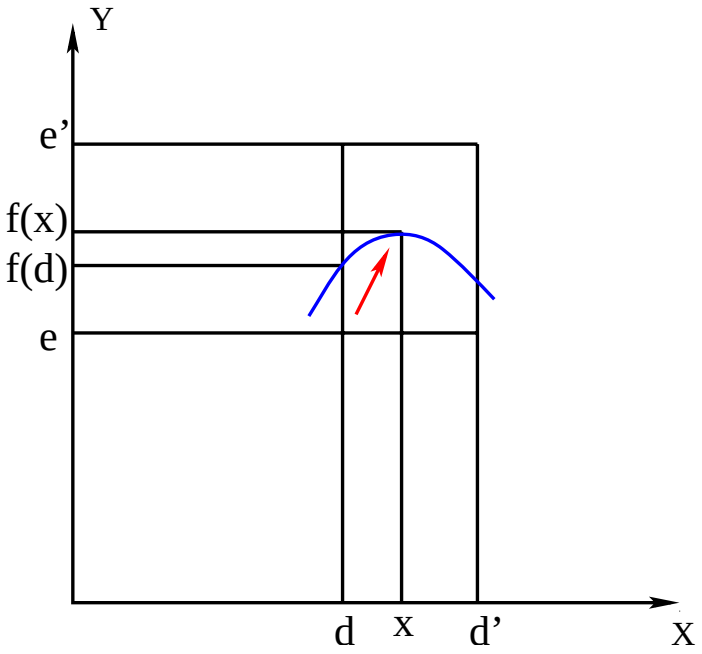


**Figure 2.1:** The computation of $f(x)$.

some modulus function $m : \mathbb{N} \to \mathbb{N}$ of continuity in the sense that, for $n \in \mathbb{N}$ and $x, y \in [0,1]$, $|x - y| \leq 2^{-m(n)} \Rightarrow |f(x) - f(y)| \leq 2^{-n}$. This property allows $f$ to be approximated by its values at dyadic points. The computability and complexity of $f$ depend on the computability and complexity of $m$ and of approximations to $f$ at dyadic points.

**Proposition 2.4.4** *A real function $f : [0,1] \to \mathbb{R}$ is computable if*

(*a*) *$f$ has a computable modulus of continuity, and*

(*b*) *there exists a computable function $\psi : (\mathbb{D} \cap [0,1]) \times \mathbb{N} \to \mathbb{D}$ such that for all $d \in \mathbb{D} \cap [0,1]$ and all $n \in \mathbb{N}$, $|\psi(d,n) - f(d)| \leq 2^{-n}$.*

**Proposition 2.4.5** *A real function $f : [0,1] \to \mathbb{R}$ is polynomial-time computable if*

(*a*) *$f$ has a polynomial modulus of continuity, and*

(*b*) *the function $\psi$ in Proposition 2.4.4 is computable in time polynomial in $p(\ell(d) + n)$.*

We call the integer $n$ in $\psi(d, n)$ above the (output) precision parameter for $f$. We use $n$ instead of $\log n$ for the complexity measure in Proposition 2.4.5 since we actually require the error to be within $2^{-n}$ instead of within $n^{-1}$.

$NC$ functions can be defined based on the properties in Proposition 2.4.5:

**Definition 2.4.6** *Suppose $i \geq 0$. A function $f : [0,1] \to \mathbb{R}$ is $NC^i$ computable if and only if*

(a) *$f$ has a polynomial modulus, and*

(b) *There exists an $NC^i$ circuit family $\{C_k\}$ such that for any integers $m, n > 0$ and any $d \in \mathbb{D}_m \cap [0,1]$, $C_{\langle n,m \rangle}$ outputs a dyadic rational number $e$ such that $|e - f(d)| \leq 2^{-n}$.*

We remark that computable and polynomial-time computable functions can also be defined using the left cut representation. For example, a function $f : [0,1] \to \mathbb{R}$ is polynomial-time computable by Proposition 2.4.5, if and only if there exists a polynomial-time oracle machine $M$ such that for any oracle $\phi$ that binary converges to a real number $x \in [0,1]$, the set $\{d \in \mathbb{D} : M^\phi(d) \text{ accepts}\}$ is a general left cut of $f(x)$. Indeed, it might be more convenient to define functions of higher complexity (e.g., $NP$, $\Sigma_2^P$) using the left cut representation, since classes of sets are more common than classes of functions on these levels.

**Definition 2.4.7** *A real function $f : [0,1] \to \mathbb{R}$ is said to be* computable in $NP$ time*, or simply an NP-real function, if there exists a nondeterministic polynomial-time oracle machine $M$ such that for any oracle $\phi$ that binary converges to a real number $x \in [0,1]$, the set $\{d \in \mathbb{D} : M^\phi(d) \text{ accepts}\}$ is a general left cut of $f(x)$.*

**Example 2.4.8** *Show that the function $f(x) = exp(x) = e^x$ on $[0,1]$ is polynomial-time computable.*

*Proof.* The proof is similar to that of Example 2.4.2. We expand $e^x$ to a power series:

$$e^x = \sum_{n=0}^{\infty} \frac{x^k}{k!}.$$

Then to approximate $e^x$ with a required precision $2^{-n}$, we approximate the first $n$ terms in the power series and sum them up, and ignore other terms. □

The above definitions are certainly extensible to functions from $[0,1]$ or $[0,1]^2$ to $\mathbb{R}$ or $\mathbb{C}$, and to other complexity classes. We will present the definitions when we need them in later chapters. Suppose $\mathcal{C}$ is a complexity class such as $P, L$ and $NC$. We use $\mathcal{C}_{C[0,1]}$ to denote the class of all $\mathcal{C}$ computable real functions defined on $[0,1]$. Complexity classes of real functions have hierarchies similar to that of discrete complexity classes. For example, Ko [40] proved that $P = NP$ iff $P_{C[0,1]} = NP_{C[0,1]}$, and Hoover [37] proved that $NC = P$ iff $NC_{C[0,1]} = P_{C[0,1]}$. Ko [40] gave many important results on characterization of continuous problems. We present one of them below as a flavor.

**Proposition 2.4.9** (*a*) *A real number $x$ is a left NP-real number if and only if there is a polynomial-time computable real function $f : [0,1] \to \mathbb{R}$ such that $x = \max_{0 \le t \le 1} f(t)$.*

(*b*) *A real number $x$ is a left $\Pi_2^P$-real number if and only if there is an NP-real function $f : [0,1] \to \mathbb{R}$ such that $x = \min_{0 \le t \le 1} f(t)$.*

## 2.5 Computable Sets in $\mathbb{R}^2$

Sets in $\mathbb{R}^2$ are an important subject in geometry, as well as real and complex analysis. The properties of these sets themselves or of functions defined on them are widely studied, in both mathematics and computer science. The basic questions to be studied here are the complexity of operations defined on sets which have polynomial-time computability structures.

Chou and Ko [18] introduced a few notions of polynomial-time computable sets in $\mathbb{R}^2$, which we list below.

Given an oracle Turing machine $M$ and a set $S \subseteq \mathbb{R}^2$, for $n \in \mathbb{N}$, we define an *error set* $E_n(M)$ as the set of all $\mathbf{z} \in \mathbb{R}^2$ having a Cauchy function representation $\langle \phi, \psi \rangle$ such that $M^{\phi,\psi}(n) \neq \chi_S(\mathbf{z})$, where $\chi_S(\mathbf{z})$ is the characteristic function defined on $\mathbb{R}^2$ such that $\chi_S(\mathbf{z}) = 1$ if $\mathbf{z} \in S$, and 0 otherwise.

**Definition 2.5.1** (*a*) *We say a set* $S \subseteq \mathbb{R}^2$ *is* polynomial-time approximable (*or simply* $P$-approximable) *if there exists a polynomial-time oracle Turing machine* $M$ *such that for any input* $n$*, the error set* $E_n(M)$ *has size* $\mu^*(E_n(M)) \leq 2^{-n}$*, where* $\mu^*$ *is the outer Lebesgue measure.*

(*b*) *We say a set* $S \subseteq \mathbb{R}^2$ *is* polynomial-time recognizable (*or simply* $P$-recognizable) *if there exists a polynomial-time oracle Turing machine* $M$ *such that* $M^{\phi,\psi}(n) = \chi_S(\mathbf{z})$ *whenever* $\langle \phi, \psi \rangle$ *represents a point* $\mathbf{z}$ *whose distance to* $\Gamma_S$ *is* $> 2^{-n}$ (*where* $\Gamma_S$ *is the boundary of* $S$)*; i.e.,* $E_n(M) \subseteq \{\mathbf{z} : \mathrm{dist}(\mathbf{z}, \Gamma_S) \leq 2^{-n}\}$.

The above two concepts are not equivalent by Chou and Ko [18]: there exists a set $S$ that is $P$-recognizable but not $P$-approximable; the polynomial-time probabilistic class $BPP$ will be identical to $P$ if all $P$-approximable sets are $P$-recognizable. On the other hand, while there exists a $P$-recognizable set $S$ whose measure is not recursive, the measure of a $P$-approximable set is polynomial-time computable relative to an oracle in $\#P$.

Because the Lebesgue measure $\mu$ is consistent with *area* of the traditional sense (e.g., if $S$ is a rectangle, $\mu(S)$ is the area of $S$), hereafter we use *area* and the Lebesgue measure interchangeably. Note that there exists a set $S \subseteq \mathbb{R}^2$ that is not measurable. However, in this dissertation we mainly consider measurable sets $S$.

In computational complex analysis, there is another basic object: a bounded, simply connected region; that is, a bounded, connected open set with no holes (or,

equivalently, whose complement is connected). A particularly interesting class of simply connected regions are those whose boundaries are Jordan curves, that is, simple closed curves. In mathematics, a Jordan curve $\Gamma$ may be represented as the image of a continuous function $f$ from $[0,1]$ to $\mathbb{R}^2$, where $f$ is 1-1 on $[0,1)$ and $f(0) = f(1)$. Note that $f$ is continuous, but not required to be smooth. In our setting, we further require that $f$ is computable. If $f$ is polynomial-time computable, we say the curve $\Gamma$ is polynomial-time computable. A bounded, simply connected region whose boundary is a polynomial-time computable Jordan curve, called a *$P$-computable Jordan domain*, is often the main object in our studies.

Let $S$ be a $P$-computable Jordan domain. In order to study functions $f$ defined on the domain $S$, we need to define the notion of computability and complexity of such functions. Intuitively, the computability of a complex function $f : S \to \mathbb{C}$ can be defined as follows: $f$ is computable on $S$ if there exists an oracle Turing machine $M$ such that, given any two oracles $\phi$ and $\psi$ that represent a complex number $\mathbf{z} \in S$ and an input $n \in \mathbb{N}$, $M$ outputs a dyadic point $\mathbf{d}$ such that $|\mathbf{d} - f(\mathbf{z})| \leq 2^{-n}$. This definition, however, appears too strict. We observe that, in general, the function $f$ may not have a continuous extension on the boundary $\partial S$ of $S$, hence the oracle Turing machine $M$ for $f$ may not halt on some $\mathbf{z} \in \partial S$, and, in addition, it may require extra time to compute the correct value of $f(\mathbf{z})$ when $\mathbf{z}$ is very close to the boundary $\partial S$.

Our approach to resolve this issue is to be less restrictive and allow the machine $M$ that computes $f$ to make errors, while the errors are required to be under control. This notion is a generalization of the notion of polynomial-time recognizable sets introduced in Chou and Ko [18].

**Definition 2.5.2** $(a)$ *Let $S$ be a bounded, simply connected domain whose boundary $\partial S$ is a computable Jordan curve. A function $f : S \to \mathbb{C}$ is* computable on domain $S$ *if*

*there exists an oracle Turing machine $M$ such that for any oracles $(\phi, \psi)$ representing a complex number $\mathbf{z} \in S$, $|M^{\phi,\psi}(n) - f(\mathbf{z})| \leq 2^{-n}$ for all inputs $n > 0$ whenever $\delta(\mathbf{z}, \partial S) > 2^{-n}$.*

(*b*) *Furthermore, $f$ is* polynomial-time computable on the domain $S$ *if $f$ is computable on $S$ by an oracle Turing machine that operates in polynomial time.*

The following definition is the *NC* version of this approach, which will be used later.

**Definition 2.5.3** *Let $S \subseteq \mathbb{C}$ be a bounded domain. Suppose $i \geq 0$. A complex function $f : S \to \mathbb{C}$ is $NC^i$ computable if the following conditions hold:*

(*a*) *$f$ has a polynomial modulus. More precisely, there exists a polynomial function $p : \mathbb{N} \to \mathbb{N}$ such that for all $m, n \in \mathbb{N}$ and all $\mathbf{z}_1, \mathbf{z}_2 \in S$, if $\delta(\mathbf{z}_1, \partial S) \geq 2^{-p(m)}$ and $|\mathbf{z}_1 - \mathbf{z}_2| \leq 2^{-p(m+n)}$, then $|f(\mathbf{z}_1) - f(\mathbf{z}_2)| \leq 2^{-n}$.*

(*b*) *There exists an $NC^i$ circuit family $\{C_n\}$ such that for any integers $m, n, k > 0$ and a dyadic point $\mathbf{d} \in S \cap \mathbb{D}_m^2$, $C_{\langle m,n+k\rangle}$ outputs a dyadic point $\mathbf{e}$ such that $|\mathbf{e} - f(\mathbf{d})| \leq 2^{-n}$ provided that $\delta(\mathbf{d}, \partial S) \geq 2^{-p(k)}$, where $p$ is the polynomial in (a).*

## 2.6 Other Models

We briefly explain other computational models for continuous functions here. Some of these models also use the Turing machine as a basic model; other models are based on different computational devices, such as the real RAM. Each model aims at a specific domain of problems.

**Turing machine-based Models** The Turing machine model is considered the basic model for computability and complexity theory in the discrete world. Many models of continuous computation use the Turing machine model as the basic model.

One of these models is the *Type-2 Theory of Effectivity* (TTE) (see, e.g., Weihrauch [77; 78]), which extends the ordinary (Type-1) computability and complexity theory. The machines used in this model, called *Type-2 machines*, transfer infinite sequences to infinite sequences, where infinite sequences (of, e.g., decimal fractions) are used as names of real numbers. Therefore, a Type-2 machine $M$ is used to compute a real function $f$ by computing a name $v$ of $f(x)$ when a name $u$ of a real number $x$ is presented. Type-2 machines, which never halt, are extensions of Turing machines. This model is widely used in computable analysis by many researchers(see, e.g., [5; 64; 86; 87]). Weihrauch's book "Computable Analysis" [78] collects many results in analysis obtained by applying this model. TTE (as well as the model used by Pour-El and Richards [63]) is essentially equivalent to the oracle Turing machine model in the sense that the class of computable real functions remains the same in both models.

There is another Turing machine-based model by Papadimitriou [58]. This model is very close to the oracle Turing machine model. In order to compute a function $f(x)$ in this model, instead of using a Cauchy function $\phi$ of $x$ as an oracle, it sometimes uses the code of a Turing machine that computes $x$ as the input.

**Real-RAM-based models** The real-RAM model [55] is a generalization of the Random Access Machine (RAM) (see, e.g., [22]). The basic objects in a real-RAM are integers and real numbers. In other words, besides registers for integers, a real-RAM also contains registers for real numbers. A real-RAM can be viewed as a Turing machine which can store a real number in each tape cell. In a real-RAM, two real numbers can be added, multiplied, divided, or compared in one step. This is quite different from the Turing machine-based models, in which it is undecidable whether two real numbers are equal (see, e.g., Ko [40, Theorem 2.5]).

It turns out that many fundamental computable functions with respect to the Turing machine-based models are not computable in the real-RAM model; for exam-

ple, neither the square root function $\sqrt{x}$ nor the exponential function $e^x$ on $\mathbb{R}$ are computable in a real-RAM. On the other hand, a real-RAM can also compute some functions that are incomputable on a Turing machine based model. A computable function in a Turing machine based model must be continuous, while a computable function in a real-RAM model can be discontinuous.

Real-RAMs are unrealistic since it is impossible to identify an arbitrary real number by a finite amount of information and therefore it is impossible to handle such a number in a finite amount of time or with a finite amount of space by any physical devices. On the other hand, however, using *floating-point numbers* the real-RAM computations can be "approximated" reasonably to some extent. Treating real numbers simply as objects also make the real-RAM model "easy to use". The real-RAM model is widely used in some theories such as algebraic complexity (see, e.g., [14]).

**Information-based complexity theory.** Information-based complexity theory, established by Traub et al. [73], is a complexity theory that distinguishes the notion of *information complexity* from the notion of *computational complexity* of a problem.

Informally speaking, for any given problem with an input function $f$, a finite amount of information $N = N(f)$ can be collected. There might be infinitely many functions $g$ in the domain of the input functions that have the same set of information $N(g) = N$. The radius $r(N)$ of $N$ is defined as the radius of the minimum *ball* that contains all these functions $g$, where the measure of the ball depends on the problem. A basic theorem in this theory is that the information $N$ is strong enough to solve a problem with an error bounded by $\epsilon$ with respect to all functions in the domain (disregarding the computational complexity) if and only if $r(N) < \epsilon$. Using this theorem, for a specific problem, the *optimum* and the *optimal algorithm* with respect to this optimal information can be computed.

The computational model used in this theory is close to the real-RAM model but with some new, non-conventional operations. It can be thought of as an extended real-RAM model with oracles, where the information $N(f)$ is from the oracles.

**BSS model.** Blum et al. [10] introduced a new theory of $NP$-completeness based on the $BSS$ model, an algebraic computational model close to the real-RAM model. Just like the real-RAM model, the BSS model is more suitable for algebraic computational problems. However, this theory differs from the classic algebraic complexity theory with two new features. First, the machine in this theory can prestore a finite number of fixed real numbers, or in other words, the machine is parameterized. This leads to uncountably many machines since there are uncountably many real numbers, while in the Turing machine-based theories, there are only a countable number of machines. Second, a nondeterministic node of a machine in this theory can guess a real number with infinite precision, while in Turing machine-based theories, a nondeterministic node can guess only a single-bit. So nondeterministic machines in this theory cannot be simulated by deterministic machines, since from a nondeterministic node, there can be an uncountable number of guesses.

To be more precise, a (deterministic) machine in this theory is a real-RAM with arithmetic operations and the compare-and-branch operation that compares two real numbers and then branches to new instructions according to the comparison result. Let $\mathbb{R}^\infty$ denote the set of all infinite sequences of real numbers with only a finite number of nonzeros. A sequence $\bar{x} = (x_1, x_2, \cdots) \in \mathbb{R}^\infty$ is of *size* n, denoted $\ell(\bar{x}) = n$, if $x_n \neq 0$ and for all $k > n$, $x_k = 0$. A *decision problem* $A$ is a subset of $\mathbb{R}^\infty$. The complexity class $P_\mathbb{R}$ (or, $NP_\mathbb{R}$) in this theory is the class of decision problems that are accepted by deterministic (or, respectively, nondeterministic) polynomial-time machines in this theory, where the running time for deciding whether a sequence $\bar{x} \in \mathbb{R}^\infty$ is in a decision problem $A$ is expressed in terms of $\ell(\bar{x})$. The question of

whether $P_{\mathbb{R}} = NP_{\mathbb{R}}$ in this theory is not known to be equivalent to the classical $P$ versus $NP$ question. The best known result is $P = PSPACE \Rightarrow P_{\mathbb{R}} = NP_{\mathbb{R}}$ [10].

**Exact Geometric Computation (EGC).** Yap [80; 81] proposed a computational framework, called the *exact geometric computation* (*EGC* for short), in which the numerical and algebraic worlds can co-exist and complement each other. In other words, EGC is a framework that merges the Turing machine-based models and real-RAM-style models, and the computation in EGC is "semi-numerical". We have seen that a big difference between the Turing machine-based models and the real-RAM-based models is their behavior on the *zero problem*, which asks whether a given real number equals zero. Namely, this problem is undecidable in the Turing machine-based model, and is a trivial question in the real-RAM-based models. An important feature of the EGC theory is that the zero problem in this theory is "partially approximable" in the sense that for a class of expressions $E$ involving some operators and integers, whether $E = 0$ or not can be decided as follows: a *zero bound* $B(E) > 0$ can be computed so that if $E \neq 0$, then $|E| > B(E)$; then an approximation $\widetilde{E}$ to $E$ is computed so that $|\widetilde{E} - E| < B(E)/2$, and "$E = 0$" is declared if $|\widetilde{E}| < B(E)/2$.

# Chapter 3

# The Logarithm and Square Root Functions on a Complex Domain

## 3.1 Introduction

Finding single-valued, analytic branches of a multi-valued function defined on a simply connected domain[1] $S$ is a fundamental problem in computational complex analysis (cf. Henrici [35]). Many well-known functions, such as the logarithm function and the square root function, are multi-valued functions and have (probably infinitely) many single-valued, analytic branches on certain simply connected domains. For some domains $S$ (e.g., the complex plane with the positive half real axis removed), it is easy to find these branches for these functions. However, in general, it is not an easy task. In particular, the computational complexity of finding single-valued, analytic branches often depends on the topological structure and complexity of the domain $S$.

To be more precise, let $S$ and $T$ be two bounded, simply connected domains in the two-dimensional plane such that $S$ and $T$ are disjoint (i.e., $S \cap T = \emptyset$). The logarithm function defined on $S \times T$ is the multi-valued function $\log(\mathbf{z} - \mathbf{a})$ that

[1]Recall that a domain is a nonempty, connected, open subset of the complex plane $\mathbb{C}$.

satisfies $e^{\log(\mathbf{z}-\mathbf{a})} = \mathbf{z} - \mathbf{a}$ for $\mathbf{z} \in S$ and $\mathbf{a} \in T$. It is well known that this function $\log(\mathbf{z} - \mathbf{a})$ has infinitely many single-valued, analytic branches. Nevertheless, for a fixed pair $\langle \mathbf{z}_0, \mathbf{a}_0 \rangle \in S \times T$, the value $f_1(\mathbf{z} - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a}_0)$ remains the same for an arbitrary single-valued, analytic branch $f_1$ of $\log(\mathbf{z} - \mathbf{a})$ (cf. Henrici [35]). Therefore, to compute all single-valued, analytic branches of $\log(\mathbf{z} - \mathbf{a})$, we only need to compute $f_1(\mathbf{z}-\mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a}_0)$ for an arbitrary single-valued, analytic branch $f_1$ of $\log(\mathbf{z} - \mathbf{a})$. The value of $f_1(\mathbf{z}-\mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a}_0)$ depends on the relative locations of $\mathbf{z}$ and $\mathbf{z}_0$ in $S$ and that of $\mathbf{a}$ and $\mathbf{a}_0$ in $T$. For instance, assume that $\mathbf{z} - \mathbf{a}_0$ and $\mathbf{z}_0 - \mathbf{a}_0$ have the same argument (i.e., there exists an angle $\theta \in [0, 2\pi)$ such that $\mathbf{z} - \mathbf{a}_0 = |\mathbf{z} - \mathbf{a}_0|e^{i\theta}$ and $\mathbf{z}_0 - \mathbf{a}_0 = |\mathbf{z}_0 - \mathbf{a}_0|e^{i\theta}$). Then, the imaginary part of $f_1(\mathbf{z} - \mathbf{a}_0) - f_1(\mathbf{z}_0 - \mathbf{a}_0)$ depends on how many times a path in $S$ from $\mathbf{z}_0$ to $\mathbf{z}$ must wind around the point $\mathbf{a}_0$. Thus, the problem of finding single-valued, analytic branches of $\log(\mathbf{z} - \mathbf{a})$ is closely related to the problem of computing the winding numbers in a domain $S$, and is dependent on the computational complexity of the domain $S$ itself.

In this chapter, we study the problem of finding the single-valued, analytic branches of the logarithm function $\log(\mathbf{z} - \mathbf{a})$ and the square root problem $\sqrt{\mathbf{z} - \mathbf{a}}$, in the context of complexity theory of real functions of Ko and Friedman [42]. In this theory, we use the oracle Turing machine as the basic computational model, and define the complexity of a real function in terms of precisions of the output values of the functions under consideration. In particular, we focus on simply connected domains $S$ in the complex plane $\mathbb{C}$ whose boundaries $\partial S$ are polynomial-time computable Jordan curves, and use notions in discrete complexity theory to characterize the computational complexity of these functions defined on such a domain.

Note that $f_1(\mathbf{z}-\mathbf{a}) - f_1(\mathbf{z}_0-\mathbf{a}_0) = (f_1(\mathbf{z}-\mathbf{a}) - f_1(\mathbf{z}_0-\mathbf{a})) + (f_1(\mathbf{z}_0-\mathbf{a}) - f_1(\mathbf{z}_0-\mathbf{a}_0))$ for an arbitrary single-valued, analytic branch $f_1$ of $\log(\mathbf{z} - \mathbf{a})$, and any method of computing $f_1(\mathbf{z} - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a})$ also applies to $f_1(\mathbf{z}_0 - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a}_0)$. Therefore,

as far as the computational complexity is concerned, we only need to study how to compute $f_1(\mathbf{z}-\mathbf{a}) - f_1(\mathbf{z}_0-\mathbf{a})$, for a fixed $\mathbf{z}_0 \in S \cup \partial S$. Under this setting, we can formally state our problem as follows. (In the following, we let $\overline{S}$ denote the closure $S \cup \partial S$ of $S$, and let $S - \mathbf{a}$ denote the domain $\{\mathbf{w}-\mathbf{a} \mid \mathbf{w} \in S\}$.)

> Logarithm Problem: Let $S$ be a bounded, simply connected domain whose boundary $\partial S$ is a polynomial-time computable Jordan curve. Let $\mathbf{z}_0$ be a fixed point in $\overline{S}$. Given two points $\mathbf{z} \in S$ and $\mathbf{a} \in \mathbb{C} - \overline{S}$, compute $f_1(\mathbf{z}-\mathbf{a}) - f_1(\mathbf{z}_0-\mathbf{a})$, where $f_1$ is an arbitrary single-valued, analytic branch of $\log \mathbf{z}$ on domain $S - \mathbf{a}$.

Similarly, the problem of computing single-valued, analytic branches of $\sqrt{\mathbf{z}-\mathbf{a}}$ can be formulated in the following form.

> Square Root Problem: Let $S$ be a bounded, simply connected domain whose boundary $\partial S$ is a polynomial-time computable Jordan curve. Let $\mathbf{z}_0$ be a fixed point in $\overline{S}$. Given two points $\mathbf{z} \in S$ and $\mathbf{a} \in \mathbb{C} - \overline{S}$, compute $f_1(\mathbf{z}-\mathbf{a})/f_1(\mathbf{z}_0-\mathbf{a})$, where $f_1$ is an arbitrary single-valued, analytic branch of $\sqrt{\mathbf{z}}$ on domain $S - \mathbf{a}$.

We observe that, in the above two problems, when $\mathbf{z}$ or $\mathbf{a}$ is on or very close to the boundary $\partial S$, the function value, or its approximation, may be incomputable or very hard to compute. This is because the underlying function cannot, in general, be extended beyond $\partial S$, and because the problem of determining whether a point $\mathbf{z}$ is on a computable curve $\Gamma$ is undecidable [44].

Thus, we follow the approach discussed in Section 2.5 to allow the underlying oracle Turing machine to have errors when $\mathbf{z}$ or $\mathbf{a}$ is close to $\partial S$. More precisely, when the oracle Turing machine computes the value of a function at $\mathbf{z}$ and $\mathbf{a}$ up to precision

$2^{-n}$, it may make errors if either $\mathbf{z}$ or $\mathbf{a}$ is close to $\partial S$ within a distance of $\leq 2^{-n}$ (cf. Definition 2.5.2).

Based on this less restrictive model, we are able to characterize the computational complexity of the logarithm problem and the square root problem with the counting complexity classes $\#P$, $\oplus P$ and $MP$ (also known as *MidBitP* in literature). (See Section 2 for the definitions of these complexity classes.) Our main results can be stated in terms of the relations between class $P$ and these counting complexity classes:

**(1)** The logarithm problem is polynomial-time solvable if and only if $FP = \#P$.

**(2)** If $P = MP$, then the square root problem is polynomial-time solvable.

**(3)** If $P \neq \oplus P$, then the square root problem is not polynomial-time solvable.

Result (1) reflects the intuition, as discussed earlier, that computing single-valued, analytic branches of the logarithm function is closely related to the computation of winding numbers, which is known to have complexity $P^{\#P}$ (see Chou and Ko [18]). It is interesting to point out, though, that the technique of computing winding numbers by integration along the boundary $\partial S$, as used in Chou and Ko [18], is not sufficient for our problem here. Instead, our algorithm for the logarithm and square root problems is more involved and makes use of many properties of simply connected domains and Jordan curves. We include a detailed description of this method in Section 2.

## 3.2 An Algorithm for Continuous Argument Functions

In this section, we will present a method for computing *continuous argument functions*, which is a critical step for the logarithm problem. Throughout this section, let $S$ be a bounded, simply connected domain whose boundary $\partial S$ is a polynomial-time

computable Jordan curve. We assume that $\partial S$ is represented by a polynomial-time computable function $f : [0, 1] \to \mathbb{C}$ and also use $f$ to denote the image of $f$ (i.e., $\partial S$).

As explained in Section 3.1, the logarithm problem is polynomial-time solvable if and only if $f_1(\mathbf{z} - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a})$ is polynomial-time computable, where $\mathbf{z}_0$ is a fixed point in $\overline{S}$, $\mathbf{z} \in S$, $\mathbf{a} \in \mathbb{C} - \overline{S}$ and $f_1$ is an arbitrary single-valued, analytic branch of $\log \mathbf{z}$ on domain $S - \mathbf{a}$. Furthermore, we note that we may assume that the second input point $\mathbf{a}$ is in a bounded domain $T \subseteq \mathbb{C} - \overline{S}$. To see this, suppose all $\mathbf{z} \in S$ have $|\mathbf{z}| < m$ for some $m > 0$. Then for any point $\mathbf{a}$ with $|\mathbf{a}| \geq m + 1$, the imaginary part of $f_1(\mathbf{z} - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a})$ is a directed angle $\alpha \in (-\pi, \pi)$ from half line $\overrightarrow{\mathbf{az}_0}$ to half line $\overrightarrow{\mathbf{az}}$. It follows that $f_1(\mathbf{z} - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a})$ is trivially polynomial-time computable. This observation also applies to the square root problem. Thus, as far as the time complexity is concerned, we may assume that $|\mathbf{a}| < m + 1$; that is, let $T = \{\mathbf{a} \in \mathbb{C} - \overline{S} : |\mathbf{a}| < m + 1\}$ and we study the complexity of computing $f_1(\mathbf{z} - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a})$ for $\mathbf{z} \in S$ and $\mathbf{a} \in T$. In other words, we may assume that both $\mathbf{z}$ and $\mathbf{a}$ to the logarithm problem are bounded.

Now we define continuous argument functions. Let $arg(\mathbf{z})$ denote the arguments of $\mathbf{z} \in \mathbb{C} - \{\langle 0, 0\rangle\}$; that is, $arg$ is a multi-valued function from $\mathbb{C} - \{\langle 0, 0\rangle\}$ to $\mathbb{R}$ such that $\mathbf{z} = |\mathbf{z}|e^{arg(\mathbf{z})i}$ (note that we also treat $arg(\mathbf{z})$ as a set of real numbers). We define a function $h_S : \overline{S} \times T \to \mathbb{R}$ such that (i) for any fixed point $\mathbf{a} \in T$, $h_S(\mathbf{z}, \mathbf{a})$ is continuous, (ii) $h_S(f(0), \mathbf{a}) = 0$ for any $\mathbf{a} \in T$, and (iii) $2\pi \cdot h_S(\mathbf{z}, \mathbf{a})$ equals $\theta_1 - \theta_2$ for some $\theta_1 \in arg(\mathbf{z} - \mathbf{a})$ and some $\theta_2 \in arg(f(0) - \mathbf{a})$. We call this function $h_S$ *the continuous argument function* of $S$. Let $\mathbf{z}_0 = f(0)$. It is obvious that the imaginary part of $f_1(\mathbf{z} - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a})$ is equal to $2\pi \cdot h_S(\mathbf{z}, \mathbf{a})$, where $f_1$ is an arbitrary single-valued, analytic branch of $\log \mathbf{z}$ in the domain $S - \mathbf{a}$.

**Lemma 3.2.1** (*a*) *The logarithm problem on $S$ is polynomial-time solvable if and only if $h_S(\mathbf{z}, \mathbf{a})$ is polynomial-time computable.*

(*b*) *The square root problem on $S$ is polynomial-time solvable if and only if the function* $(h_S(\mathbf{z}, \mathbf{a}) \bmod 2)$ *is polynomial-time computable.*

*Proof.* Statement (*a*) holds because for any single-valued, analytic branch $f_1$ of $\log \mathbf{z}$ on $S - \mathbf{a}$, $f_1(\mathbf{z} - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a}) = \log|\mathbf{z} - \mathbf{a}| - \log|\mathbf{z}_0 - \mathbf{a}| + h_S(\mathbf{z}, \mathbf{a}) \cdot 2\pi i$ and $\log|\mathbf{z} - \mathbf{a}| - \log|\mathbf{z}_0 - \mathbf{a}|$ is polynomial-time computable.

Statement (*b*) holds because for any single-valued, analytic branch $f_1$ of $\sqrt{\mathbf{z}}$ on $S - \mathbf{a}$, $f_1(\mathbf{z} - \mathbf{a})/f_1(\mathbf{z}_0 - \mathbf{a}) = (\sqrt{|\mathbf{z} - \mathbf{a}|}/\sqrt{|\mathbf{z}_0 - \mathbf{a}|})e^{h_S(\mathbf{z},\mathbf{a})\pi i}$, and $e^{2\pi i} = 1$ and $\sqrt{|\mathbf{z} - \mathbf{a}|}/\sqrt{|\mathbf{z}_0 - \mathbf{a}|}$ is polynomial-time computable. □

Now we consider how to compute $h_S(\mathbf{z}, \mathbf{a})$ for $\mathbf{z} \in \overline{S}$ and $\mathbf{a} \in T$. For a point $\mathbf{z} = f(t)$ on $\partial S$, we can compute $h_S(\mathbf{z}, \mathbf{a})$ by computing the integration of $1/(\mathbf{z} - \mathbf{a})$ over the curve $f([0, t])$ (note that the indefinite integral of $1/(\mathbf{z}-\mathbf{a})$ is $\log(\mathbf{z} - \mathbf{a})+C$), which is a generalization of Theorem 6.4 in Chou and Ko [18]. We include the details in Section 3.2.1.

For points $\mathbf{z} \in S$, the situation is more complicated. Intuitively, we can compute $h_S(\mathbf{z}, \mathbf{a})$ as follows (see Figure 3.1 shown in Section 3.2):

(a) Let $L$ be the half line starting from $\mathbf{z}$ going in the direction from $\mathbf{a}$ to $\mathbf{z}$. Then, find a real number $t_0 \in [0, 1]$ such that $f(t_0)$ lies on $L$, and the line segment $\overline{\mathbf{z}f(t_0)}$ lies entirely in $\overline{S}$.

(b) Compute $h_S(f(t_0), \mathbf{a})$ by integration; then let $h_S(\mathbf{z}, \mathbf{a}) = h_S(f(t_0), \mathbf{a})$.

However, we observe that, in general, Step (a) is hard to implement, since, in general, the point $t_0$ may be a nonrecursive real number[2]. Therefore, we cannot

[2]For instance, let $g : [0, 1] \to \mathbb{R}$ be a polynomial-time computable function such that $g(x) \geq 0$ for all $x \in [0, 1]$ and all roots of $g$ are nonrecursive (cf. Corollary 4.3 of Ko [40]). Assume that $f : [0, 1] \to \mathbb{C}$ defines a Jordan curve and $f(t) = \langle 2t, g(2t)\rangle$ when $0 \leq t \leq 1/2$, then all intersection points of $f$ and the half line $\overrightarrow{\langle -1, 0\rangle\langle 1, 0\rangle}$ are nonrecursive.

follow Step (a) directly. On the other hand, what we really need is just the value of $h_S(f(t_0), \mathbf{a})$ instead of the value of $t_0$. We will present, in Section 3.2.2, an algorithm that explores the curve $\partial S$ to find some candidates $t$ for $t_0$, and uses these candidate points to find the correct value of $h_S(f(t_0), \mathbf{a})$.

Before we describe our algorithm for $h_S$, we observe that to compute $h_S(\mathbf{z}, \mathbf{a})$, we only need to focus on dyadic points $\mathbf{z}$ and $\mathbf{a}$ in $\mathbb{D}^2$. Let $\mathbf{z}_1$, $\mathbf{z}_2$ be two points in $\overline{S}$ and $\mathbf{a}_1$, $\mathbf{a}_2$ two points in $\mathbb{C} - \overline{S}$ such that $\delta(\mathbf{z}_1, \partial S) > 2^{-n}$, $|\mathbf{z}_2 - \mathbf{z}_1| < 2^{-(n+1)}$, $\delta(\mathbf{a}_1, \partial S) > 2^{-n}$ and $|\mathbf{a}_2 - \mathbf{a}_1| < 2^{-(n+1)}$. Then,

$$\begin{aligned} |h_S(\mathbf{z}_1, \mathbf{a}_1) - h_S(\mathbf{z}_2, \mathbf{a}_2)| &\leq |h_S(\mathbf{z}_1, \mathbf{a}_1) - h_S(\mathbf{z}_1, \mathbf{a}_2)| + |h_S(\mathbf{z}_1, \mathbf{a}_2) - h_S(\mathbf{z}_2, \mathbf{a}_2)| \\ &< 1/6 + 1/6 = 1/3, \end{aligned}$$

i.e.,

$$h_S(\mathbf{z}_1, \mathbf{a}_1) - 1/3 < h_S(\mathbf{z}_2, \mathbf{a}_2) < h_S(\mathbf{z}_1, \mathbf{a}_1) + 1/3.$$

Thus, we can compute $h_S(\mathbf{z}_2, \mathbf{a}_2)$ easily from any values of $arg(\mathbf{z}_2, \mathbf{a}_2)$ and $arg(f(0), \mathbf{a}_2)$ and the value of $h_S(\mathbf{z}_1, \mathbf{a}_1)$. It follows that, as far as polynomial-time computability is concerned, we only need to make sure that the machine $M$ that computes $h_S$ works for all dyadic points $\mathbf{z}$ and $\mathbf{a}$.

### 3.2.1 A Simple Case

In this subsection, we present an algorithm for $h_S(\mathbf{z}, \mathbf{a})$ for the special case where $\mathbf{z} = f(t)$ for some $t \in [0, 1]$ (which is given as an input). The algorithm is based on the integration technique, which has been used by Chou and Ko [18] in the study of winding numbers.

We will consider a more general problem that does not require the closed curve $\partial S$ to be simple. Let $f : [0, 1] \to \mathbb{C}$ be a polynomial-time computable function that represents a closed curve $\Gamma$ (not necessarily simple). Then, there is a continuous

function $g_f : [0,1] \times (\mathbb{C} - \Gamma) \to \mathbb{R}$, called a *continuous argument function through a curve*, that satisfies two conditions:

(1) $g_f(0, \mathbf{a}) = 0$ for all $\mathbf{a} \in \mathbb{C} - \Gamma$.

(2) $g_f(t, \mathbf{a}) \cdot 2\pi \in \{\theta_1 - \theta_0 : \theta_1 \in arg(f(t), \mathbf{a}), \theta_0 \in arg(f(0), \mathbf{a})\}$ for all $t \in [0,1]$ and $\mathbf{a} \in \mathbb{C} - \Gamma$.

Note that $g_f(1, \mathbf{a})$ is just the winding number[3] of $\mathbf{a}$ with respect to $\Gamma$; that is, $g_f$ is an extension of the notion of winding numbers. In addition, in the case that $\Gamma$ is simple and $S$ is the interior of $\Gamma$, $g_f(t, \mathbf{a}) = h_S(f(t), \mathbf{a})$ for $\mathbf{a} \in \mathbb{C} - \overline{S}$.

**Lemma 3.2.2** (Chou and Ko [18]) *Let $f : [0,1] \to \mathbb{C}$ be a polynomial-time computable function that represents an arc $\Gamma$, and $\mathbf{d}$ a dyadic point in $\mathbb{C} - \Gamma$. Assume that there exists an $\epsilon > 0$ such that $\delta(\mathbf{d}, \Gamma) > \epsilon$. Then there exists a polynomial-time oracle Turing machine that computes the function $g_f(\beta, \mathbf{d}) - g_f(\alpha, \mathbf{d})$, whenever $\alpha$ and $\beta$ satisfy the condition that for all $t_1, t_2 \in [\alpha, \beta]$ $(\subseteq [0,1])$, $|f(t_1) - f(t_2)| < \epsilon$.*

The proof of Lemma 3.2.2 uses the following fact: Based on the conditions given in the lemma, the curve $f([\alpha, \beta])$ and the line segment $\overline{f(\alpha)f(\beta)}$ are *homotopic* with respect to $\mathbb{C} - \{\mathbf{d}\}$. Therefore, the integrals of any analytic function on $\mathbb{C} - \{\mathbf{d}\}$ over the curve $f([\alpha, \beta])$ and the line segment $\overline{f(\alpha)f(\beta)}$ are equal to each other (See Cauchy's theorem in Chapter 4 of Henrici [35]). In other words, the closed curve consisting of $f([\alpha, \beta])$ and $\overline{f(\beta)f(\alpha)}$ does not *circle around* the point $\mathbf{d}$.

**Theorem 3.2.3** *Let $f : [0,1] \to \mathbb{C}$ be a polynomial-time computable function that represents a closed curve $\Gamma$. Then there exists an oracle Turing machine that computes $g_f$ in polynomial time using a function $G$ in $\#P$ as an oracle.*

[3]Informally, the winding number of a closed curve $\Lambda$ around a point $\mathbf{w}$ not on $\Lambda$ is the number of times the curve $\Lambda$ circles around $\mathbf{w}$.

*Proof.* This is a slight generalization of Theorem 6.4 in Chou and Ko [18]. The inputs are $t \in [0,1]$, $\mathbf{a} \in \mathbb{C} - \Gamma$ and an integer $n$. By the earlier discussion, we may assume that $\mathbf{a}$ is a dyadic point and $t$ is a dyadic rational. Then, we need to construct a polynomial-time oracle Turing machine that, on these inputs and an oracle $G \in \#P$, outputs a number $d \in \mathbb{D}_n$ that satisfies $|d - g_f(t, \mathbf{a})| \leq 2^{-n}$ whenever $\delta(\mathbf{a}, \Gamma) > 2^{-n}$.

Assume that $f$ has a polynomial modulus function $p$. That is, $|f(t_1) - f(t_2)| \leq 2^{-n}$ for any two numbers $t_1, t_2 \in [0,1]$ satisfying $|t_1 - t_2| \leq 2^{-p(n)}$. From Lemma 3.2.2, there exists a polynomial-time Turing machine $M$ that, on inputs $\mathbf{a}, \alpha, \beta, n, k$, computes an approximation to $g_f(\beta, \mathbf{a}) - g_f(\alpha, \mathbf{a})$ within error $2^{-k}$, whenever $0 \leq \beta - \alpha \leq 2^{-p(n)}$ and $\delta(\mathbf{a}, \Gamma) > 2^{-n}$.

We now define a polynomial-time computable function $F : \{0,1\}^* \times (\mathbb{D} \cap [0,1]) \times ((\mathbb{C} - \Gamma) \cap \mathbb{D}^2) \to \mathbb{N}$ as follows: For any string $w \in \{0,1\}^*$ with $\ell(w) = p(n)$, let $i_w$ be the nonnegative integer $\leq 2^{p(n)} - 1$ whose $p(n)$-bit binary representation is $w$. For any integer $i \leq 2^{p(n)}$, let $s_i = i \cdot 2^{-p(n)}$. Let $F(w, t, \mathbf{a})$ be the function computed by the following algorithm:

(1) If $\ell(w) \neq p(n)$ for any $n \geq 0$, then output 0.

(2) If $\ell(w) = p(n)$ and $s_{i_w} \geq t$, then output 0.

(3) If $\ell(w) = p(n)$ and $s_{i_w+1} < t$ then simulate $M$ on input $(\mathbf{a}, s_{i_w}, s_{i_w+1}, n, n+p(n))$ to get a dyadic rational $e_w \in \mathbb{D}_{p(n)+n}$ such that $|e_w - (g_f(s_{i_w+1}, \mathbf{a}) - g_f(s_{i_w}, \mathbf{a}))| \leq 2^{-(p(n)+n)}$, and output $j_w = (e_w + 1) \cdot 2^{p(n)+n}$.

(4) If $\ell(w) = p(n)$ and $s_{i_w} < t \leq s_{i_w+1}$ then simulate $M$ on input $(\mathbf{a}, s_{i_w}, t, n, n + p(n))$ to get a dyadic rational $e_w \in \mathbb{D}_{p(n)+n}$ such that $|e_w - (g_f(t, \mathbf{a}) - g_f(s_{i_w}, \mathbf{a}))| \leq 2^{-(p(n)+n)}$, and output $j_w = (e_w + 1) \cdot 2^{p(n)+n}$.

We note that $e_w$ must satisfy $-1/4 \leq e_w \leq 1/4$, and so $e_w + 1 > 0$. Thus, $F(w, t, \mathbf{a})$ is always nonnegative.

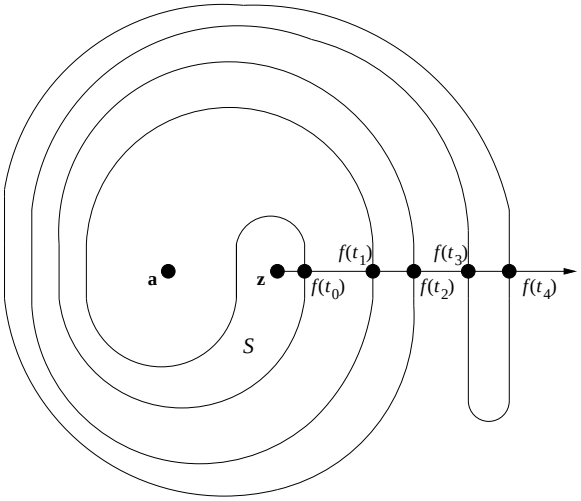


**Figure 3.1:** $L$ and $\partial S$ have an odd number of intersections.

Now, define $G(0^m, t, \mathbf{a}) = \sum_{\ell(w)=m} F(w, t, \mathbf{a})$. Then, by Theorem 2.3.2 (c), $G$ is a function in $\#P$, since $F$ is polynomial-time computable. Furthermore, $G(0^{p(n)}, t, \mathbf{a}) \cdot 2^{-(p(n)+n)} - \lceil t \cdot 2^{p(n)} \rceil$ is an approximation to $g_f(t, \mathbf{a})$ with an error $\leq 2^{-n}$, where $\lceil x \rceil$ is the ceiling function. Therefore, $g_f$ is polynomial-time computable relative to the oracle $G \in \#P$. □

### 3.2.2 The Algorithm

Let $L$ be the half line starting from $\mathbf{z}$ going in the direction from $\mathbf{a}$ to $\mathbf{z}$. We say $L$ and $\partial S$ intersect *non-degenerately* if (1)$L \cap \partial S$ contains only finitely many points, and (2) if $f(t)$ is in $L \cap \partial S$, then $\partial S$ actually crosses $L$ at $f(t)$. It is easy to see that, if $L$ and $\partial S$ intersect non-degenerately, then $L \cap \partial S$ contains an odd number of points $f(t_0), f(t_1), \ldots, f(t_{2m})$ (see Figure 3.1). Furthermore, the values $g_f(t_i, \mathbf{a})$ can be canceled out in the following sense.

**Proposition 3.2.4** *Let $f(t_0)$, $f(t_1)$, $\cdots$, $f(t_{2m})$, where $m \geq 0$, be all points in $L \cap \partial S$ in the order of their distances away from $\mathbf{z}$, with $f(t_0)$ being the closest one to $\mathbf{z}$. Then, they satisfy the following properties:*

(1) *For any $1 \leq i \leq m$, $\overline{f(t_{2i-1})f(t_{2i})}$ lies entirely in $\overline{S}$, and $g_f(t_{2i-1}, \mathbf{a}) = g_f(t_{2i}, \mathbf{a})$.*

(2) *For any* $1 \leq i \leq m$, $f$ *crosses* $L$ *from opposite directions at points* $f(t_{2i-1})$ *and* $f(t_{2i})$.

(3) $h_S(\mathbf{z}, \mathbf{a}) = g_f(t_0, \mathbf{a})$.

Our basic idea of computing $h_S(\mathbf{z}, \mathbf{a})$ is as follows. Assume that $L$ and $\partial S$ intersect non-degenerately. Then, we can get values $g_f(t_i, \mathbf{a})$, for all $0 \leq i \leq 2m$, by applying the integration algorithm over the curve in Section 3.2.1 to compute $g_f(t, \mathbf{a})$ over all points $f(t)$ at which $f$ crosses $L$. Note that, in the above computation, we will get all values of $t_0, t_1, \ldots, t_{2m}$, but cannot tell which one is $t_0$ (because the distance between $f(t_0)$ and $f(t_1)$ could be too small for us to tell which one is closer to $\mathbf{z}$). However, we can still obtain the value of $h_S(\mathbf{z}, \mathbf{a})$ as follows:

(1) Let $\Delta$ be an integer such that $g_f(t_i, \mathbf{a}) + \Delta \geq 0$ for all $0 \leq i \leq 2m$.

(2) For each $0 \leq i \leq 2m$, let $sgn_i$ be 1 or $-1$ according to the direction in which $f$ crosses $L$ at $f(t_i)$ (e.g., $+1$ if $f$ crosses $L$ counterclockwise, and $-1$ if $f$ crosses $L$ clockwise).

(3) Let $h_S(\mathbf{z}, \mathbf{a}) = \left| \sum_{i=0}^{2m} (sgn_i \cdot (g_f(t_i, \mathbf{a}) + \Delta)) \right| - \Delta$.

That is, we use the factor $sgn_i$ to cancel out the values of $g_f(t_{2i-1}, \mathbf{a}) + \Delta$ and $g_f(t_{2i}, \mathbf{a}) + \Delta$ in the summation of (3), and the only one left is $g_f(t_0, \mathbf{a}) + \Delta$ (yet we do not know what the value $t_0$ is). We use the extra term $\Delta$ so that $g_f(t_0, \mathbf{a})$ can be extracted from the absolute value of $sgn_0 \cdot (g_f(t_0, \mathbf{a}) + \Delta)$; that is, because $g_f(t_0, \mathbf{a}) + \Delta \geq 0$,

$$g_f(t_0, \mathbf{a}) = |g_f(t_0, \mathbf{a}) + \Delta| - \Delta = |sgn_0 \cdot (g_f(t_0, \mathbf{a}) + \Delta)| - \Delta.$$

There are some technical problems with these ideas. First, since we can only approximate $\partial S$, we may not be able to compute the intersections of $L$ and $\partial S$

correctly. Second, $L$ and $\partial S$ may not intersect non-degenerately. That is, one of the following situations may occur:

(1) $L \cap \partial S$ contains infinitely many points (e.g., the curve $\partial S$ may cross $L$ infinitely many times); or

(2) $f(t)$ is in $L \cap \partial S$ for some $t \in [0,1]$, but $f$ does not cross $L$ at $f(t)$.

In the following, we describe a method to approximate intersections of $\partial S$ and $L$ that will solve the above problems. The main idea is to use a piecewise linear curve $f_n$ that approximates $f$ and apply the integration method on $f_n$. By a careful analysis, we can show that this computation is still correct.

First, based on the discussion in the beginning of this section, we assume that $\mathbf{z}$ and $\mathbf{a}$ are dyadic points. (Thus, we can tell whether a dyadic point lies on $L$ or not.) Next, let $M$ be an oracle Turing machine that computes $f$ in time $p$ for some polynomial $p$. It follows that $p$ is a modulus function of $f$. Let $n$ be an integer such that $\delta(\mathbf{z}, \partial S) > 2^{-n}$ and $\delta(\mathbf{a}, \partial S) > 2^{-n}$. For each $0 \leq i \leq 2^{p(2n)}$, let $s_i = i \cdot 2^{-p(2n)}$ and $\mathbf{z}_i = M^{s_i}(2n)$. Then, for any $0 \leq i \leq 2^{p(2n)} - 1$ and any $t \in [s_i, s_{i+1}]$, we have $|f(t) - \mathbf{z}_i| \leq 2^{-2n}$. Let $f_n$ be the piecewise linear function with breakpoints $f_n(s_i) = \mathbf{z}_i$, for $i = 0, \ldots, 2^{p(2n)}$, and $\Gamma_n$ be the image of $f_n$ on $[0,1]$. Then, $\Gamma_n$ is an approximation of $\partial S$ within an error $2^{-2n}$. Note that $\Gamma_n$ is not necessarily simple.

We now define a function $sgn_{f_n}$ which assigns value $+1$, $-1$ or $0$ to each directed line segment of $\Gamma_n$ according to whether it crosses $L$ counterclockwise, or crosses $L$ clockwise, or does not cross $L$, respectively. More precisely, let $L'$ be the straight-line that contains $L$. Then, $L'$ divides the plane into two half planes, the one on the left of $L$ is denoted $S_1$, and the other half plane plus $L'$ is denoted $S_2$. We define

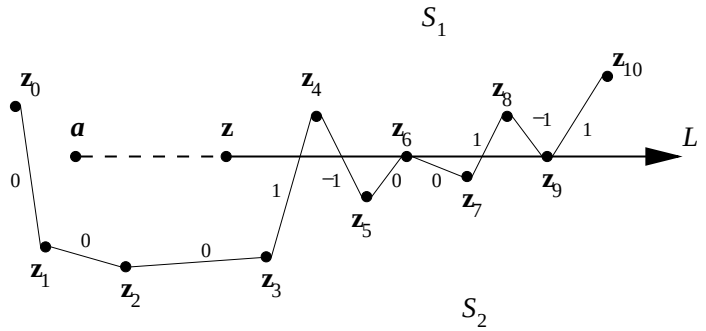


**Figure 3.2:** The function $sgn_{f_n}$ (points $\mathbf{z}_6$ and $\mathbf{z}_9$ lie on $L$).

$sgn_{f_n} : \{1, 2, \ldots, 2^{p(2n)}\} \to \{-1, 0, 1\}$ as follows (see Figure 3.2):

$$sgn_{f_n}(i) = \begin{cases} 1, & \text{if } \mathbf{z}_{i-1} \in S_2 \text{ and } \mathbf{z}_i \in S_1 \text{ and } \overline{\mathbf{z}_{i-1}\mathbf{z}_i} \cap L \neq \emptyset, \\ -1, & \text{if } \mathbf{z}_{i-1} \in S_1 \text{ and } \mathbf{z}_i \in S_2 \text{ and } \overline{\mathbf{z}_{i-1}\mathbf{z}_i} \cap L \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases}$$

When $sgn_{f_n}(i)$ is not zero, there is a unique $s'_i \in [s_{i-1}, s_i]$ such that $\mathbf{z}'_i := f_n(s'_i) \in \overline{\mathbf{z}_{i-1}\mathbf{z}_i} \cap L$; we call $\mathbf{z}'_i$ an intersection point of $f_n([s_{i-1}, s_i])$ and $L$, while if $sgn_{f_n}(j) = 0$ (including the case $f_n([s_{j-1}, s_j]) \subset L$), we say there is no intersection point of $f_n([s_{j-1}, s_j])$ and $L$. Note that the points $\mathbf{z}'_i$'s may overlap each other but $s'_i$'s may not.

**Lemma 3.2.5** *Assume that $\partial S$ is polynomial-time computable. Then the functions $\phi_1(n, i) = sgn_{f_n}(i)$, $\phi_2(n, i) = s'_i$ and $\phi_3(n, i) = \mathbf{z}'_i$ (if they exist) are polynomial-time computable.*

*Proof.* We note that all points $\mathbf{a}$, $\mathbf{z}$ and $\mathbf{z}_i$ are dyadic points. Thus, $sgn_{f_n}(i)$ can be computed by exact arithmetic. When $sgn_{f_n}(i) \neq 0$, $s'_i$ and $\mathbf{z}'_i$ are not necessarily dyadic but must be rational. We can compute, for any $k \geq p(2n)$, a dyadic $d_k \in \mathbb{D}_k$, such that $|d_k - s'_i| \leq 2^{-k}$, which implies that $|f_n(d_k) - f_n(s'_i)| \leq 2^{-(2n+k-p(2n))}$. □

For simplicity, we may assume that $f_n(0) = f(0)$ (this can be achieved by, for example, transforming the whole plane so that $f(0)$ becomes the origin).

**Theorem 3.2.6** *Assume hereafter $s'_i = s_i$ if $sgn_{f_n}(i) = 0$. Then we have*

$$|h_s(\mathbf{z}, \mathbf{a})| = \Big| \sum_{i=1}^{2^{p(2n)}} sgn_{f_n}(i) \cdot g_{f_n}(s'_i, \mathbf{a}) \Big| = |g_{f_n}(s'_{i_0}, \mathbf{a})|, \tag{3.1}$$

*where $1 \le i_0 \le 2^{p(2n)}$, $|sgn_{f_n}(i_0)| = 1$ and the number of line segments $f_n([s_{i-1}, s_i])$ $(1 \le i \le 2^{p(2n)})$ with $sgn_{f_n}(i) = sgn_{f_n}(i_0)$ is exactly one more than those with $sgn_{f_n}(i) = -sgn_{f_n}(i_0)$.*

To prove Theorem 3.2.6, we need to show that

(1) In the sum of the middle term of equation (3.1), all values but one of $g_{f_n}(s'_i, \mathbf{a})$ are canceled out. (Since the curve $\Gamma_n$ is only an approximation to $\partial S$, it is not necessarily a simple curve and hence this fact does not follow from Proposition 3.2.4 immediately.)

(2) The remaining term after cancellation is $\pm h_S(\mathbf{z}, \mathbf{a})$.

To do this, we divide the interval $[0, 1]$ into a finite number of subintervals, and examine each subinterval separately. First, define $\mathbf{w}_1 \in S_1$, $\mathbf{w}_2 \in S_2$ to be the two points which have distance $2^{-(2n-1)}$ from $\mathbf{z}$ such that $\mathbf{z}$ lies on the line segment $\overline{\mathbf{w}_1\mathbf{w}_2}$ and $\overline{\mathbf{w}_1\mathbf{w}_2}$ is perpendicular to $L$ (see Figure 3.3). For each $i = 1, 2$, define a half line $L_i$ that starts at $\mathbf{w}_i$ and runs parallel to $L$. Call the domain between the two half lines $L_1$ and $L_2$ and the line segment $\overline{\mathbf{w}_1\mathbf{w}_2}$ (including the boundary) the *crossing zone*. We let $Z$ denote the crossing zone.

Without loss of generality, assume that $\mathbf{z}_0 = f(0)$ and the crossing zone $Z$ are on different sides of line $\overleftrightarrow{\mathbf{w}_1\mathbf{w}_2}$.[4] We call an interval $[b, c] \subseteq [0, 1]$ a *crossing interval* if $[b, c]$ is a maximal interval with the following properties: (i) $f(b) \in L_i$ and $f(c) \in L_{3-i}$ for $i = 1$ or 2, and (ii) $f([b, c])$ lies entirely in the crossing zone $Z$. (By "maximal"

[4]If it is not the case, we can pick a point $s \in [0, 1]$ such that $f(s)$ and the crossing zone $Z$ are on different sides of line $\overleftrightarrow{\mathbf{w}_1\mathbf{w}_2}$, and use the function $f_1(t) = f(s + t \bmod 1)$ on $[0, 1]$ to represent $\partial S$.

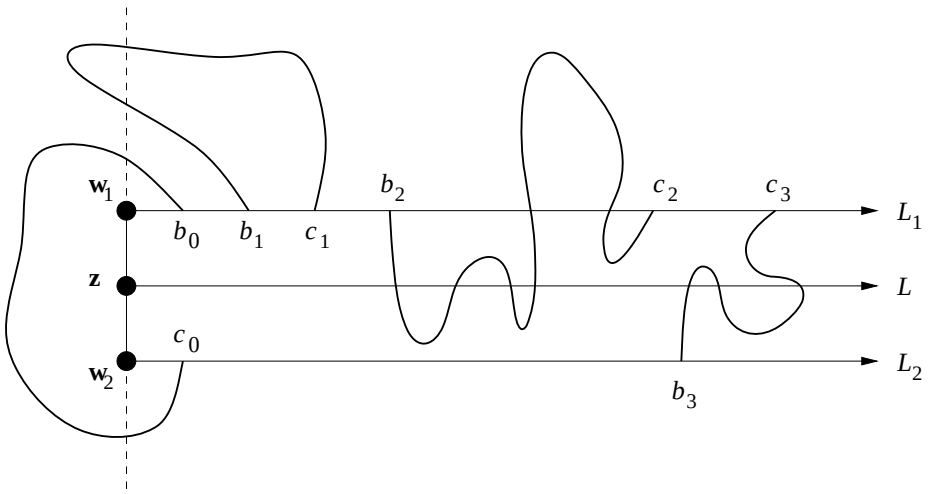


**Figure 3.3:** Curve $\partial S$ on three different types of intervals.

we mean that all intervals $[b', c']$ that properly contain $[b, c]$ do not satisfy both (i) and (ii).) We call an interval $[b, c]$ a *non-crossing interval* if (1) $f(b) \in L_1 \cup L_2$ or $b = 0$, and $f(c) \in L_1 \cup L_2$ or $c = 1$, and (2) $f((b, c))$ lies entirely outside of the crossing zone $Z$ and intersects the line $\overleftrightarrow{\mathbf{w}_1\mathbf{w}_2}$. Note that there are only a finite number of crossing intervals and non-crossing intervals since, due to the definitions and the facts that $\partial S$ has a modulus of continuity, the length of each such interval is bounded below. If we remove all crossing intervals and non-crossing intervals from $[0, 1]$, the remainder is the union of a finite number of intervals. We call each such interval a *semi-crossing interval*. A semi-crossing interval $[b, c]$ satisfies the following conditions: (i) both $f(b)$ and $f(c)$ are in $L_i$ for $i = 1$ or 2, (ii) if $f(b) \in L_i$ for $i = 1$ or 2, then $f([b, c]) \cap L_{3-i} = \emptyset$, and (iii) $f([b, c])$ does not intersect line $\overleftrightarrow{\mathbf{w}_1\mathbf{w}_2}$. Figure 3.3 shows the curve $\partial S$ on these intervals, where $[b_0, c_0]$ and $[b_1, c_1]$ are two non-crossing intervals, $[b_2, c_2]$ is a semi-crossing interval, and $[b_3, c_3]$ is a crossing interval.

**Lemma 3.2.7** *Let $\mathbf{z}_1, \mathbf{z}_2$ be two points in $\overline{S} \cap L$. Assume that there is a path $\pi$ from $\mathbf{z}_1$ to $\mathbf{z}_2$ that lies in $\overline{S} \cap Z$. Then, $h_S(\mathbf{z}_1, \mathbf{a}) = h_S(\mathbf{z}_2, \mathbf{a})$.*

*Proof.* We note that if $\pi \subseteq Z$, then it cannot go around the point $\mathbf{a}$, and so $h_S(\mathbf{z}_1, \mathbf{a}) = h_S(\mathbf{z}_2, \mathbf{a})$. □

**Lemma 3.2.8** *Let $[b, c]$ be a non-crossing interval. Then, $sgn_{f_n}(i) = 0$ for all $s_i \in [b, c]$, and so $\sum_{s_i \in [b,c]} sgn_{f_n}(i) \cdot g_{f_n}(s'_i, \mathbf{a}) = 0$.*

*Proof.* We note that the distance between $L_1$ and $L$ (and the distance between $L_2$ and $L$) is $2^{-(2n-1)}$, and that our approximation $f_n$ and $f$ has distance at most $2^{-2n}$. Therefore, $f_n([b, c])$ cannot touch the half line $L$. □

**Lemma 3.2.9** *Let $[b, c]$ be a semi-crossing interval. Then, there are an even number of intersection points $f_n(r_1), f_n(r_2), \cdots, f_n(r_{2m})$ in $f_n([b, c]) \cap L$, and $g_{f_n}$ has the same value $g_{f_n}(r_1, \mathbf{a})$ at all $r_i$'s. These values all cancel out after considering the crossing directions; that is, $\sum_{s_i \in [b,c]} sgn_{f_n}(i) \cdot g_{f_n}(s'_i, \mathbf{a}) = 0$.*

*Proof.* Without loss of generality, assume that $f(b) \in L_1$. Then, $f([b, c]) \cap L_2 = \emptyset$. Since the curve $f([b, c])$ does not intersect the line $\overleftrightarrow{\mathbf{w}_1\mathbf{w}_2}$, $f([b, c])$ cannot "circle around" the point $\mathbf{a}$. Furthermore, since $\mathbf{a}$ is at least $2 \cdot 2^{-n}$ away from the line $\overleftrightarrow{\mathbf{w}_1\mathbf{w}_2}$ and the curve $f_n([b, c])$ is a $2^{-2n}$ approximation to $f([b, c])$, $f_n([b, c])$ cannot circle around the point $\mathbf{a}$ either. Thus, the values $g_{f_n}(r_j, \mathbf{a})$ are all equal.

Now, assume that there are $k$ such intersection points $f_n(r_1), f_n(r_2), \ldots, f_n(r_k)$, with $b < r_1 < r_2 < \cdots < r_k < c$. Then, we observe that $f_n$ at $r_1$ must go from $S_1$ to $S_2$, because $f(b) \in L_1$. Since $f([b, c])$ cannot cross $L_2$, $f_n([b, c])$ cannot go back to domain $S_1$ without passing through $L$ (i.e., it cannot go around point $\mathbf{a}$). Therefore, $f_n$ at $r_2$ must go from $S_2$ to $S_1$. From this observation, we see that there must be an even number of intersection points (i.e., $k = 2m$ for some $m \geq 0$), and the crossing direction of $f_n$ at $r_{2j-1}$ is the opposite of that of $f_n$ at $r_{2j}$, for each $j = 1, \ldots, m$. □

For each crossing interval $[b, c]$, define $sgn_{[b,c]} = +1$ if $f(b) \in L_2$ and $f(c) \in L_1$; and $sgn_{[b,c]} = -1$ otherwise.

**Lemma 3.2.10** *Let $[b, c]$ be a crossing interval.*

(*a*) *There exists at least one point* $t \in [b,c]$ *such that* $f(t) \in L$. *For any two such numbers* $t_1, t_2 \in [b,c]$ *with* $f(t_1), f(t_2) \in L$, $g_f(t_1, \mathbf{a}) = g_f(t_2, \mathbf{a})$.

(*b*) *There are an odd number of intersection points* $f_n(r_0), f_n(r_1), \ldots, f_n(r_{2m})$ *in* $f_n([b,c]) \cap L$, *and they all have the same value* $g_{f_n}(r_i, \mathbf{a})$ *as* $g_f(t, \mathbf{a})$. *All but one of these values cancel out after considering the crossing directions; that is,* $\sum_{s_i \in [b,c]} sgn_{f_n}(i) \cdot g_{f_n}(s_i, \mathbf{a}) = sgn_{[b,c]} g_f(t, \mathbf{a})$.

*Proof.* (a): Similar to the proof of Lemma 3.2.9, the curve $f([b,c])$ cannot go around the point $\mathbf{a}$, and so the values $g_f(t,a)$ at all intersection points are the same. This fact can also be proved by Lemma 3.2.7: The curve $f([b,c])$ lies entirely in the crossing zone $Z$, and so it is a path connecting all these intersection points. So, by Lemma 3.2.7, they all have the same value $g_f(t, \mathbf{a})$.

(b): The proof for this part is similar to that of Lemma 3.2.9. That is, if $f_n(r_0)$, $f_n(r_1), \ldots, f_n(r_k)$ are all the intersection points with $b < r_0 < r_1 < \cdots < r_k < c$, then $g_{f_n}(r_j, \mathbf{a})$ must all have the same value as $g_f(t, \mathbf{a})$, and they must cross $L$ alternately in opposite directions; for example, if $f(b) \in L_1$, then $f_n$ at $r_0$ goes from $S_1$ to $S_2$, and $f_n$ at $r_1$ goes from $S_2$ to $S_1$, etc. It follows that there are an odd number of intersection points. Thus, all terms, except the first one, in the sum $\sum_{s_i \in [b,c]} sgn_{f_n} \cdot g_{f_n}(s_i, \mathbf{a})$ cancel out. The remaining one has the same direction as $sgn_{[b,c]}$ and the same value as $g_f(t, \mathbf{a})$. □

Now we can prove Theorem 3.2.6.

*Proof.* Let $\mathcal{C}$ denote the collection of all crossing intervals. For each interval $[b,c] \in \mathcal{C}$, choose a representative $t_{b,c} \in [b,c]$ with $f(t_{b,c}) \in L$. From the above three lemmas, we see that

$$\sum_{i=1}^{2^{p(2n)}} sgn_{f_n}(i) \cdot g_{f_n}(s_i', \mathbf{a}) = \sum_{[b,c] \in \mathcal{C}} sgn_{[b,c]} g_f(t_{b,c}, \mathbf{a}).$$

It remains to show that the above sum cancels to a single term which equals $\pm h_S(\mathbf{z}, \mathbf{a})$.

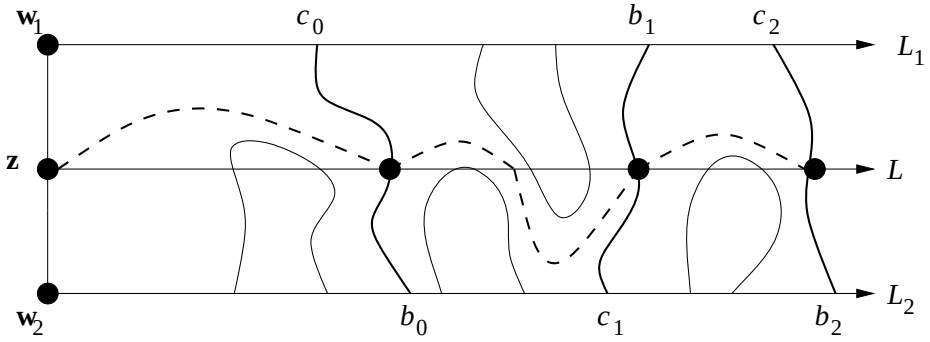


**Figure 3.4:** Connecting points $\mathbf{z}$, $f(t_0)$, $f(t_1)$ and $f(t_2)$ within $Z$.

We first observe that, since $f$ is a simple curve, any two curves defined by $f$ on any two crossing intervals do not cross each other. In other words, we can arrange all crossing intervals $I_0 = [b_0, c_0]$, $I_1 = [b_1, c_1], \ldots,$ $I_k = [b_k, c_k]$ according to the distances of $f(I_j)$ from $\mathbf{z}$ $(0 \leq j \leq k)$, with $f(I_0)$ being the closest one to $\mathbf{z}$. For each $j = 0, \ldots, k$, let $t_j$ be the representative of intersection points in $f([b_j, c_j]) \cap L$; i.e., $t_j = t_{b_j, c_j}$.

Now, since $[b_0, c_0]$ is the closest crossing interval to $\mathbf{z}$, there is a path $\pi_0$ from $\mathbf{z}$ to $f(t_0)$ that lies entirely in $\overline{S} \cap Z$. By Lemma 3.2.7, we have $g_f(t_0, \mathbf{a}) = h_S(f(t_0), \mathbf{a}) = h_S(\mathbf{z}, \mathbf{a})$ (see Figure 3.4).

If $k = 0$, we are done. If $k \geq 1$, then there is a path $\pi_1$ from $f(t_0)$ to $f(t_1)$ that does not touch $\partial S$ (except at $f(t_0)$ and $f(t_1)$) and lies in $Z$. Since $\pi_0$ lies in $\overline{S}$, we see that $\pi_1$ lies in $\mathbb{C} - S$. Thus, if we go from $f(t_1)$ and cross the curve $f([b_1, c_1])$, we must enter domain $S$. This means that there must be a third crossing interval $[b_2, c_2]$, and there is a path $\pi_2$ from $f(t_1)$ to $f(t_2)$ that does not touch $\partial S$ (except at $f(t_1)$ and $f(t_2)$) and lies in $Z$. This path $\pi_2$ is in $\overline{S} \cap Z$, and so, by Lemma 3.2.7, $g_f(t_1, \mathbf{a}) = g_f(t_2, \mathbf{a})$.

Furthermore, we claim that $sgn_{[b_1, c_1]} = -sgn_{[b_2, c_2]}$. To see this, we first assume that $b_1 < c_1 < b_2 < c_2$. We note that $\pi_2$ divides $S$ into two simply connected domains. One of them has the boundary $f([0, t_1]) \cup \pi_2 \cup f([t_2, 1])$, which contains the points $f(b_1)$ and $f(c_2)$. This means that $f(b_1)$ and $f(c_2)$ must be in the same half line $L_i$

for $i = 1$ or 2, and so $sgn_{[b_1,c_1]}$ must equal $-sgn_{[b_2,c_2]}$. Similarly, if $b_2 < c_2 < b_1 < c_1$, then we can see that both $f(b_2)$ and $f(c_1)$ must be in the same half line $L_i$ for some $i = 1$ or 2, and it also holds that $sgn_{[b_1,c_1]} = -sgn_{[b_2,c_2]}$.

Repeating the above argument, we see that there are an odd number of crossing intervals (that is, $k = 2m$ for some integer $m \geq 0$), and $sgn_{[b_{2i-1},c_{2i-1}]}g_f(t_{2i-1}, \mathbf{a}) = -sgn_{[b_{2i},c_{2i}]}g_f(t_{2i}, \mathbf{a})$. We conclude that $\sum_{i=1}^{2^{p(2n)}} sgn_{f_n}(i) \cdot g_{f_n}(s'_i, \mathbf{a}) = sgn_{[b_0,c_0]}g_f(t_0, \mathbf{a}) = \pm h_S(\mathbf{z}, \mathbf{a})$. □

Based on Theorem 3.2.6, we can design an algorithm to compute $h_S(\mathbf{z}, \mathbf{a})$.

We first define a polynomial-time computable function $F : (S \cap \mathbb{D}^2) \times (T \cap \mathbb{D}^2) \times \{0,1\}^* \to \mathbb{N}$ (recall that $T = \{\mathbf{z} \in \mathbb{C} - \overline{S} : |\mathbf{z}| < m + 1\}$). Let $M$ be a polynomial-time Turing machine that, on input $(\mathbf{a}, \alpha, \beta, n)$ computes $g_{f_n}(\beta, \mathbf{a}) - g_{f_n}(\alpha, \mathbf{a})$, as given in Lemma 3.2.2. Let $K = 2^{2p(2n)+n+1}$. (Note that $K$ is not exactly the extra parameter $\Delta$ discussed in the beginning of this subsection; instead, we let $\Delta = 2^{p(2n)}$ and keep its role implicit in the following proof.) We also use the same notations as defined earlier, including $i_w$ and $s_i$. We let $F(\mathbf{z}, \mathbf{a}, w)$ be the function computed by the following algorithm.

(1) If $\ell(w) \neq 2p(2n)$ for any $n > 0$, then output 0.

(2) If $\ell(w) = 2p(2n)$ for some $n > 0$, let $w = uv$ with $\ell(u) = \ell(v)$.

  (2.1) If $i_v > i_u$, then let $e_v := 0$.

  (2.2) If $i_v < i_u$, then simulate $M$ on input $(\mathbf{a}, s_{i_v}, s_{i_v+1}, n)$ to get a dyadic rational $e_v \in \mathbb{D}_{2p(2n)+n+1}$ such that $|e_v - (g_{f_n}(s_{i_v+1}, \mathbf{a}) - g_{f_n}(s_{i_v}, \mathbf{a}))| \leq 2^{-(2p(2n)+n+1)}$.

  (2.3) If $i_v = i_u$, then compute a dyadic rational $d_u \in \mathbb{D}_{2p(2n)}$ such that $|d_u - s'_{i_u}| \leq 2^{-2p(2n)}$. Simulate $M$ on input $(\mathbf{a}, s_{i_u}, d_u, n)$ to get a dyadic rational $e_v \in \mathbb{D}_{2p(2n)+n+1}$ such that $|e_v - (g_{f_n}(d_u, \mathbf{a}) - g_{f_n}(s_{i_u}, \mathbf{a}))| \leq 2^{-(2p(2n)+n+1)}$.

(3) Output $sgn_{f_n}(i_u) \cdot (e_v + 1)K + 2K$.

It is clear that $F$ is polynomial-time computable. Furthermore, we note that, for any $v$, $-1/4 < e_v < 1/4$, and so $e_v + 1 < 2$. It follows that $F(\mathbf{z}, \mathbf{a}, w)$ is always nonnegative.

Note that for any $u$ of length $p(2n)$,

$$\sum_{\ell(v)=p(2n)} F(\mathbf{z}, \mathbf{a}, uv) = sgn_{f_n}(i_u) \cdot K \cdot \sum_{\ell(v)=p(2n)} (e_v + 1) + K \cdot 2^{p(2n)+1}$$
$$= sgn_{f_n}(i_u) \cdot K \cdot \sum_{\ell(v)=p(2n)} e_v + sgn_{f_n}(i_u) \cdot K \cdot 2^{p(2n)} + K \cdot 2^{p(2n)+1}.$$

We can verify that, for a fixed $u$, $\sum_{\ell(v)=p(2n)} e_v$ is a good approximation to $g_{f_n}(s'_{i_u}, \mathbf{a})$.

For $i_v < i_u$, let $\epsilon_v = e_v - ((g_{f_n}(s_{i_v+1}, \mathbf{a}) - g_{f_n}(s_{i_v}, \mathbf{a}))$. Also let $\epsilon_u = e_u - (g_{f_n}(d_u, \mathbf{a}) - g_{f_n}(s_{i_u}, \mathbf{a}))$, and $\epsilon'_u = g_{f_n}(s'_{i_u}, \mathbf{a}) - g_{f_n}(d_u, \mathbf{a})$.

By the proof of Lemma 3.2.5, $|f_n(d_u) - f_n(s'_{i_u})| \leq 2^{-(2n+2p(2n)-p(2n))} = 2^{-(p(2n)+2n)}$. Furthermore, note that $f_n(s'_{i_u})$ is in the crossing zone $Z$, we have $\delta(\mathbf{a}, f_n(s'_{i_u})) \geq \delta(\mathbf{a}, Z) = \delta(\mathbf{a}, \mathbf{z}) \geq \delta(\mathbf{a}, \partial S) + \delta(\mathbf{z}, \partial S) \geq 2 \cdot 2^{-n}$, and so $|\epsilon'_u| < 2^{-(p(2n)+n+2)}$. Therefore,

$$\sum_{\ell(v)=p(2n)} e_v = g_{f_n}(s'_{i_u}, \mathbf{a}) + \sum_{i_v \leq i_u} \epsilon_v + \epsilon'_u,$$

with the error

$$\left| \sum_{i_v \leq i_u} \epsilon_v + \epsilon'_u \right| \leq 2^{p(2n)} \cdot 2^{-(2p(2n)+n+1)} + 2^{-(p(2n)+n+2)} < 2^{-(p(2n)+n)}.$$

It follows that

$$\left| \Big( \sum_{\ell(v)=p(2n)} F(\mathbf{z}, \mathbf{a}, uv) - K \cdot 2^{p(2n)+1} \Big) / K - sgn_{f_n}(i_u)(g_{f_n}(s'_{i_u}, \mathbf{a}) + 2^{p(2n)}) \right| < 2^{-(p(2n)+n)}.$$

Now, we define a function $G : (S \cap \mathbb{D}^2) \times (T \cap \mathbb{D}^2) \times \{0\}^* \to \mathbb{N}$ by $G(\mathbf{z}, \mathbf{a}, 0^m) = \sum_{\ell(w)=m} F(\mathbf{z}, \mathbf{a}, w)$. Then, $G$ is in $\#P$. Moreover, we have

$$\left| (G(\mathbf{z}, \mathbf{a}, 0^{2p(2n)}) - K \cdot 2^{2p(2n)+1})/K - \sum_{\ell(u)=p(2n)} sgn_{f_n}(i_u)(g_{f_n}(s'_{i_u}, \mathbf{a}) + 2^{p(2n)}) \right|$$

$$< 2^{p(2n)} \cdot 2^{-(p(2n)+n)} = 2^{-n}.$$

From Theorem 3.2.6, we know that

$$\sum_{\ell(u)=p(2n)} sgn_{f_n}(i_u)(g_{f_n}(s'_{i_u}, \mathbf{a}) + 2^{p(2n)})$$

cancel out to have only one term left. This remaining term is equal to $\pm(h_S(\mathbf{z}, \mathbf{a}) + 2^{p(2n)})$. Since $f$ has a modulus function $p(n)$, and since $\delta(\mathbf{a}, \partial S) > 2^{-n}$, we know that $|h_S(\mathbf{z}, \mathbf{a})| < 2^{p(2n)}$ and thus $h_S(\mathbf{z}, \mathbf{a}) = |h_S(\mathbf{z}, \mathbf{a}) + 2^{p(2n)}| - 2^{p(2n)}$ (here $2^{p(2n)}$ serves as $\Delta$).

So, we can find an approximation of $h_S(\mathbf{z}, \mathbf{a})$ with an error $\leq 2^{-n}$ as follows:

(1) Ask oracle $G$ to get $G(\mathbf{z}, \mathbf{a}, 0^{2p(2n)})$.

(2) Let $e := |(G(\mathbf{z}, \mathbf{a}, 0^{2p(2n)}) - K \cdot 2^{2p(2n)+1})/K|$; output $e - 2^{p(2n)}$.

We just completed the proof of the following theorem:

**Theorem 3.2.11** *Let $S$ be a bounded, simply connected domain whose boundary $\partial S$ is a polynomial-time computable Jordan curve. Then there exists an oracle Turing machine that computes $h_S(\mathbf{z}, \mathbf{a})$ in polynomial time using a function $G$ in $\#P$ as an oracle.*

**Corollary 3.2.12** *If $FP = \#P$, then the continuous argument function problem and the logarithm problem are polynomial-time solvable.*

## 3.3 The Logarithm Problem

We have shown, in the last section, that $P^{\#P}$ is an upper bound for the complexity of computing the continuous argument functions, and hence the logarithm problem. In this section, we show that $P^{\#P}$ is also a lower bound for the logarithm problem.

**Theorem 3.3.1** *For any $G \in \#P$, there exist a bounded, simply connected domain $S$ whose boundary $\partial S$ is a Jordan curve represented by a polynomial-time computable function $f : [0,1] \to \mathbb{C}$, and three polynomial-time computable functions $\phi_i : \{0,1\}^* \to \mathbb{D}^2$, $i = 1,2,3$, such that for any $n > 0$ and any $w \in \{0,1\}^n$,*

*(a)* $\delta(\phi_i(w), \partial S) > 2^{-p(n)}$ *for some polynomial function $p$, $i = 1,2,3$.*

*(b)* $\phi_1(w) \notin S$ *and* $\phi_2(w), \phi_3(w) \in S$.

*(c)* $G(w) = h_S(\phi_3(w), \phi_1(w)) - h_S(\phi_2(w), \phi_1(w))$.

*Proof.* The construction of the Jordan curve is similar to that for the winding number problem in Chou and Ko [18]. We first describe a basic construction that will be used later. For any $n > 0$ and any set $B \subseteq \{0,1\}^n$, we construct a simply connected domain $S_B$ as, roughly, the interior of a rectangle with a strip of width $\epsilon > 0$ removed. This strip "winds" around a point $\mathbf{z}$ for $\|B\|$ times. Thus, the cardinality $\|B\|$ of set $B$ can be found from the continuous argument function about point $\mathbf{z}$.

We first define a function $g_B : [0, 3/4] \to \mathbb{C}$ that represents a curve $\Gamma_B$ which winds around a point $\mathbf{a}_B$ for $\|B\|$ times. For each integer $k$, $0 \le k \le 2^n - 1$, we let $u_k$ denote the n-bit binary representation of $k$.

(1) $g_B$ is linear on $[0, 1/4]$, with $g_B(0) = \langle -2, 0\rangle$ and $g_B(1/4) = \langle -1 - 2^{-n-1}, 0\rangle$.

(2) For each $k$ such that $0 \le k \le 2^n - 1$, let $t_k = 1/4 + k \cdot 2^{-n-1}$. If $u_k \notin B$, then $g_B$ is linear on $[t_k, t_{k+1}]$ with $g_B(t_k) = \langle -1 + (k-2) \cdot 2^{-n}, 0\rangle$ and $g_B(t_{k+1}) = \langle -1 + (k-1) \cdot 2^{-n}, 0\rangle$.

(3) For each $k$ such that $0 \le k \le 2^n - 1$, if $u_k \in B$, then $g_B$ is piecewise linear on $[t_k, t_{k+1}]$: It divides $[t_k, t_{k+1}]$ into 5 subintervals of equal length and maps them

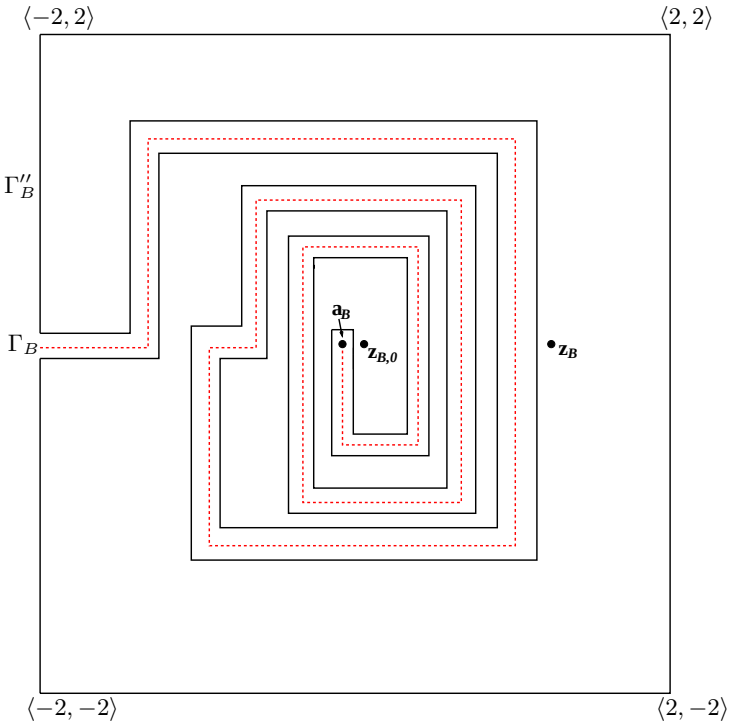


**Figure 3.5:** The domain $S_B$ for set $B = \{00, 10, 11\}$.

to the 5 consecutive line segments defined by the following breakpoints:

$$\langle -1 + (k-2)\cdot 2^{-n}, 0\rangle, \qquad \langle -1 + (k-2)\cdot 2^{-n}, 1 - (k-2)\cdot 2^{-n}\rangle,$$
$$\langle 1 - (k-2)\cdot 2^{-n}, 1 - (k-2)\cdot 2^{-n}\rangle, \qquad \langle 1 - (k-2)\cdot 2^{-n}, -1 + (k-2)\cdot 2^{-n}\rangle,$$
$$\langle -1 + (k-1)\cdot 2^{-n}, -1 + (k-2)\cdot 2^{-n}\rangle, \quad \langle -1 + (k-1)\cdot 2^{-n}, 0\rangle.$$

Next, we define a curve $\Gamma'_B$ that is the piecewise linear curve surrounding $\Gamma_B$, with distance $2^{-(n+2)}$ from it, plus the following two line segments: $\overline{\langle -2, -2\rangle, \langle -2, -2^{-(n+2)}\rangle}$ and $\overline{\langle -2, 2^{-(n+2)}\rangle, \langle -2, 2\rangle}$. Let $\Gamma''_B$ be the curve $\Gamma'_B$ plus the three line segments connecting the points $\langle -2, 2\rangle$, $\langle 2, 2\rangle$, $\langle 2, -2\rangle$ and $\langle -2, -2\rangle$. Then,$\Gamma''_B$ is a Jordan curve. Figure 3.5 shows the curves $\Gamma_B$ and $\Gamma''_B$ for set $B = \{00, 10, 11\}$ (the dotted curve denotes $\Gamma_B$ and the solid curve denotes $\Gamma''_B$). Let $S_B$ be the interior of $\Gamma''_B$.

Define $\mathbf{a}_B = \langle -2^{-(n-1)}, 0\rangle = g_B(3/4)$, $\mathbf{z}_{B,0} = \langle -2^{-(n-1)} + 2^{-(n+1)}, 0\rangle$, and $\mathbf{z}_B = \langle 1 + 2^{-(n-1)} + 2^{-(n+1)}, 0\rangle$. Then, it is easy to verify that $\mathbf{a}_B \notin S_B$, $\mathbf{z}_{B,0}, \mathbf{z}_B \in S_B$, $\delta(\mathbf{a}_B, \Gamma''_B) = \delta(\mathbf{z}_{B,0}, \Gamma''_B) = \delta(\mathbf{z}_B, \Gamma''_B) = 2^{-(n+2)}$. Furthermore, $||B|| = h_{S_B}(\mathbf{z}_B, \mathbf{a}_B) - h_{S_B}(\mathbf{z}_{B,0}, \mathbf{a}_B)$.

It is also easy to see that the function $g_B$ is polynomial-time computable if $B$ is given as an oracle. Therefore, we can define a function $f_B : [0,1] \to \mathbb{C}$ (with $f_B(1) = \langle -2, 2\rangle$) which is computable in polynomial time with $B$ as an oracle, and whose image is $\Gamma'_B$. In particular, we note that $f_B$ has a linear modulus function: $q_B(k) = n + k + c$ for some constant $c > 0$.

Now we define a function $f$ that computes the boundary $\partial S$ of domain $S$. For convenience, we will define $f$ on $[0,2]$ instead of on $[0,1]$. For any string $w \in \{0,1\}^n$, let $i_w$ be the integer whose $n$-bit binary representation is $w$. Let $a_n = 1 - 2^{-(n-1)}$ and $x_w = a_n + i_w \cdot 2^{-2n}$. Note that if $u$ is the lexicographic successor of $w$, then $x_w + 2^{-2n} = x_u$. Since $G \in \#P$, there exist a set $A \in P$ and a polynomial $q$ such that for all $w \in \{0,1\}^*$, $G(w) = ||B_w||$, where $B_w = \{u : \ell(u) = q(\ell(w)), \langle w, u\rangle \in A\}$.

For each $w \in \{0,1\}^*$, $\ell(w) = n$, we define the function $f$ on the subinterval $[x_w, x_w + 2^{-2n}]$ to be a linear transformation of $f_{B_w}$ on $[0,1]$. Let $f_1, f_2 : [0,1] \to \mathbb{R}$ be such that $f_{B_w}(t) = \langle f_1(t), f_2(t)\rangle$. Suppose $\ell(w) = n$, then $f$ on $[x_w, x_w + 2^{-2n}]$ can be defined as follows:

$$f(t) = \langle 2^{-(2n+2)} f_1(2^{2n}(t - x_w)) + 2^{-(2n+1)},\ 2^{-(2n+2)} f_2(2^{2n}(t - x_w)) + x_w + 2^{-(2n+1)}\rangle.$$

Now we define $f$ on $[1,2]$ to be a piecewise linear function mapping the interval $[1,2]$ to three line segments connecting the following four points: $\langle 0,1\rangle$, $\langle 1,1\rangle$, $\langle 1,0\rangle$ and $\langle 0,0\rangle$. It is also easy to see that $f$ on $[0,2]$ represents a Jordan curve $\Lambda$. More precisely, let $\Lambda_w$ be the image of $f$ on $[x_w, x_w + 2^{-2n}]$. Then, $\Lambda_w$ is a linear transformation of $\Gamma'_{B_w}$. Note that all $\Lambda_w$'s are connected together with $\Lambda_w$ and $\Lambda_u$ having exactly one common point, if $u$ is the lexicographic successor of $w$. Figure 3.6 shows the curve $\Lambda$.

We define $\phi_1(w)$ to be the images of $\mathbf{a}_{B_w}$, $\phi_2(w)$ the image of $\mathbf{z}_{B_w,0}$, and $\phi_3(w)$ the image of $\mathbf{z}_{B_w}$, under the above linear transformation. Let $S$ be the interior of the Jordan curve $\Lambda$. Then, according to the properties of $\mathbf{a}_{B_w}$, $\mathbf{z}_{B_w,0}$ and $\mathbf{z}_{B_w}$ and the

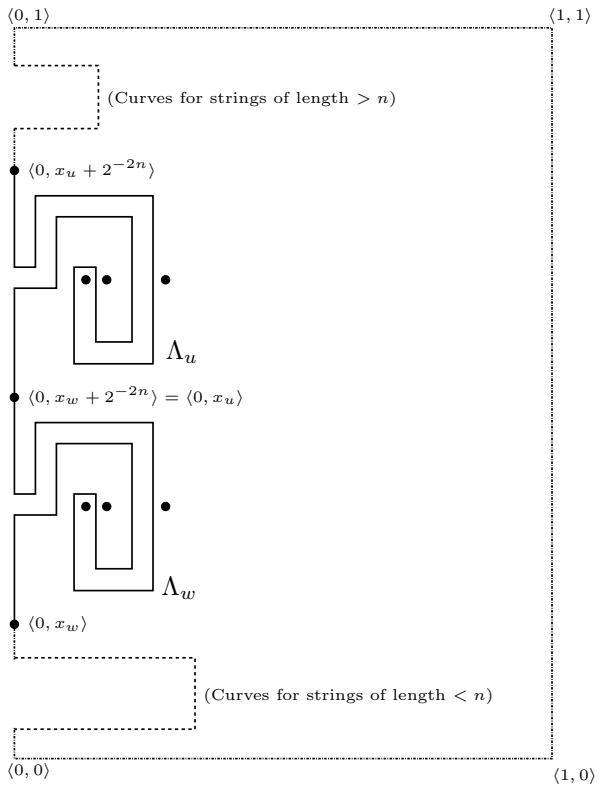


**Figure 3.6:** Function $f$ and the Jordan curve $\Lambda$.

definition of $f$, properties (a), (b) and (c) of the theorem hold.

It remains to show that $f$ is polynomial-time computable. By Proposition 2.4.5, we just need to show $f$ has a polynomial modulus of continuity. Recall that $f_{B_w}$ has a modulus function $q_{B_w}(k) = \ell(w) + k + c$.

*Claim:* If $t_1, t_2 \in [0, 2]$ and $0 \leq t_2 - t_1 \leq 2^{-(5k+6+c)}$, then $|f(t_1) - f(t_2)| \leq 2^{-k}$. (Note that if $\ell(w) \leq k + 2$, then $5k + 6 + c \geq q_{B_w}(k) + 3k + 4$.)

We prove the claim by the following case analysis:

Case 1. $t_1, t_2 \in [1, 2]$. It is obvious that $|f(t_1) - f(t_2)| \leq 3 \cdot 2^{-(5k+6+c)}$.

Case 2. $t_1, t_2 \in [a_{k+2}, 1]$. Then we must have $|f(t_i) - \langle 0, 1\rangle| \leq 2^{-(k+1)}$ for both $i = 1, 2$. Thus $|f(t_1) - f(t_2)| \leq 2^{-k}$.

Case 3. $t_1 \in [a_{k+2}, 1], t_2 \in (1, 2]$. We have $|f(t_1) - \langle 0, 1\rangle| \leq 2^{-(k+1)}$ and $|f(t_2) - \langle 0, 1\rangle| \leq 3 \cdot 2^{-(5k+6+c)}$. Thus $|f(t_1) - f(t_2)| \leq 2^{-k}$.

Case 4. $t_1, t_2 \in [x_w, x_w + 2^{-2n}]$ for some $w$ of length $n \leq k+2$. Then, $|t'_1 - t'_2| \leq 2^{-(q_{B_w}(k)+3k+4-2n)} \leq 2^{-(q_{B_w}(k)+k)}$, where $t'_i = 2^{2n}(t_i - x_w)$ for $i = 1, 2$. Therefore, $|f_{B_w}(t'_1) - f_{B_w}(t'_2)| \leq 2^{-k}$. If follows that $|f(t_1) - f(t_2)| \leq 2^{-(2n+k)} < 2^{-(k+1)}$.

Case 5. $t_1 < x_w \leq t_2$ for some $w$ of length $n \leq k+2$. Then $t_1$ must be in $[x_u, x_u + 2^{-2\ell(u)}]$, where $u$ is the lexicographic predecessor of $w$ and $x_u + 2^{-2\ell(u)} = x_w$. Then, applying Case 4 to $t_1$ and $x_w$, $x_w$ and $t_2$, we get

$$|f(t_1) - f(t_2)| \leq |f(t_1) - f(x_w)| + |f(x_w) - f(t_2)| \leq 2^{-k}.$$

This completes the proof of Claim and hence the proof of the theorem. □

**Corollary 3.3.2** *The following are equivalent:*

(*a*) $FP = \#P$.

(*b*) *For every bounded, simply connected domain $S$ whose boundary is a polynomial-time computable Jordan curve, the logarithm problem on $S$ is polynomial-time computable.*

## 3.4 The Square Root Problem

According to Lemma 3.2.1, the square root problem is polynomial-time computable if and only if the function $(h_S(\mathbf{z}, \mathbf{a}) \bmod 2)$ is polynomial-time computable. Theorem 3.2.11 shows that $h_S$ is polynomial-time computable using a function in $\#P$ as an oracle. Since we only need the value of $(h_S(\mathbf{z}, \mathbf{a}) \bmod 2)$, we actually do not need, as an oracle, the full power of a $\#P$ function. In fact, we can modify the algorithm for $h_S$ of Section 3.2.2 and use only a single bit from the oracle function $G$ to compute $(h_S(\mathbf{z}, \mathbf{a}) \bmod 2)$. Thus, the complexity of the square root problem actually can be characterized by the complexity class $MP$.

**Theorem 3.4.1** *Let $S$ be a bounded, simply connected domain whose boundary $\partial S$ is a polynomial-time computable Jordan curve. Then, the square root problem is*

*polynomial-time solvable using a function $G_1$ in $\#P$ as an oracle. In addition, the oracle machine that solves the square root problem needs only to ask the oracle for a single bit of a value of $G_1$.*

*Proof.* By Lemma 3.2.1, it suffices to show that $(h_S(\mathbf{z}, \mathbf{a}) \bmod 2)$ is polynomial-time computable with an oracle in $MP$. According to the relationship between $h_S$ and the multi-valued function $arg(\mathbf{z})$, the fractional part of $(h_S(\mathbf{z}, \mathbf{a}) \bmod 2)$, denoted by $h_{frac}(\mathbf{z}, \mathbf{a})$, is polynomial-time computable. Therefore, we only need to show that the integral part of $(h_S(\mathbf{z}, \mathbf{a}) \bmod 2)$ is polynomial-time computable with an oracle in $MP$.

Let's check the algorithm in Section 3.2.2. We designed a $\#P$ function $G$, and let $e = |(G(\mathbf{z}, \mathbf{a}, 0^{2p(2n)}) - K \cdot 2^{2p(2n)+1})/K|$, where $K = 2^{2p(2n)+n+1}$, then $e - 2^{p(2n)}$ is an approximation to $h_S(\mathbf{z}, \mathbf{a})$ with an error $\leq 2^{-n}$. Thus $((e-2^{p(2n)}) \bmod 2) = (e \bmod 2)$ is an approximation to $(h_S(\mathbf{z}, \mathbf{a}) \bmod 2)$ with an error $\leq 2^{-n}$. Now we can roughly see why the integral part of $(h_S(\mathbf{z}, \mathbf{a}) \bmod 2)$ is polynomial-time computable with an oracle in $MP$, since $e$ is polynomial-time computable with an oracle in $\#P$. However, because of the absolute value operator $|\cdot|$ in the expression for $e$, we may need, in order to compute $(e \bmod 2)$, to ask for two bits from the oracle $G(\mathbf{z}, \mathbf{a}, 0^{2p(2n)})$.

In the following, we use two tricks to solve the problem: (1) we modify function $G$ to another $\#P$ function $G_1$ so that $G_1$ only carries information of the integral part of $h_S(\mathbf{z}, \mathbf{a})$, and (2) we remove the operator $|\cdot|$ because $(|k| \bmod 2) = (k \bmod 2)$ for any integer $k$.

We first compute a number $\phi \in \mathbb{D}_{p(2n)+n+1}$ such that $|\phi - h_{frac}(\mathbf{z}, \mathbf{a})| \leq 2^{-(p(2n)+n+1)}$. We then replace each $e_v$ used in the definition of the function $F$ with $e'_v := e_v - \phi \cdot 2^{-p(2n)}$, and define a function $F_1$ similar to function $F$ except that its output is $F_1(\mathbf{z}, \mathbf{a}, w) = sgn_{f_n}(i_u) \cdot (e'_v + 1)K + 2K$ when $w = uv$ and $\ell(u) = \ell(v) = p(2n)$.

Now we define a function $G_1 : (S \cap \mathbb{D}^2) \times (T \cap \mathbb{D}^2) \times \{0\}^* \to \mathbb{N}$ by $G_1(\mathbf{z}, \mathbf{a}, 0^m) = \sum_{\ell(w)=m} F_1(\mathbf{z}, \mathbf{a}, w) + 2^{m+2}$. Following the error analysis in Section 3.2, we see that

$$|(G_1(\mathbf{z}, \mathbf{a}, 0^{2p(2n)}) - 2^{2p(2n)+2} - K \cdot 2^{2p(2n)+1})/K|$$

is an approximation to $h_S(\mathbf{z}, \mathbf{a}) - h_{frac}(\mathbf{z}, \mathbf{a}) + 2^{p(2n)}$ with an error $2^{-n} + |\phi - h_{frac}(\mathbf{z}, \mathbf{a})| < 2^{-(n-1)}$.

We note that $h_S(\mathbf{z}, \mathbf{a}) - h_{frac}(\mathbf{z}, \mathbf{a}) + 2^{p(2n)}$ is an integer, denoted $k$. Therefore, for some $sgn \in \{+1, -1\}$, we have

$$sgn \cdot k - 2^{-(n-1)} < (G_1(\mathbf{z}, \mathbf{a}, 0^{2p(2n)}) - 2^{2p(2n)+2} - K \cdot 2^{2p(2n)+1})/K < sgn \cdot k + 2^{-(n-1)},$$

and so

$$sgn \cdot k + 2^{2p(2n)+1} < G_1(\mathbf{z}, \mathbf{a}, 0^{2p(2n)})/K < sgn \cdot k + 2^{2p(2n)+1} + 2^{-(n-2)}$$

(recall that $K = 2^{2p(2n)+n+1}$). It implies that the integral part of $G_1(\mathbf{z}, \mathbf{a}, 0^{2p(2n)})/K$ is equal to $sgn \cdot k + 2^{2p(2n)+1}$ (when $n > 1$). Because

$$(h_S(\mathbf{z}, \mathbf{a}) - h_{frac}(\mathbf{z}, \mathbf{a})) \bmod 2 = k \bmod 2 = (sgn \cdot k) \bmod 2 = (sgn \cdot k + 2^{2p(2n)+1}) \bmod 2,$$

the integral part of $(h_S(\mathbf{z}, \mathbf{a}) \bmod 2)$ equals that of $(G_1(\mathbf{z}, \mathbf{a}, 0^{2p(2n)})/K) \bmod 2$, which is exactly the $(2p(2n)+n+2)$-th least significant bit of $G_1(\mathbf{z}, \mathbf{a}, 0^{2p(2n)})$. So, the square root problem can be solved by asking for only a single bit of the oracle $G_1$. □

**Corollary 3.4.2** *If $P = MP$, then the square root problem is solvable in polynomial time.*

*Remark.* It is interesting to point out that our algorithm for the square root problem does not seem to require the full power of $MP$. Indeed, the function $G_1(\mathbf{z}, \mathbf{a}, 0^{2p(2n)})$ has the special property that $G_1(\mathbf{z}, \mathbf{a}, 0^{2p(2n)})/K$ is very close to an integer $g$, and all we need is the least significant bit of the integer $g$. This seems to suggest that

the square root problem might actually be in a weaker complexity class than $P^{MP}$. Whether it can be solved in $P^{\oplus P}$ (e.g., whether the integer $g$ can be expressed as the output of a $\#P$ function) remains an open question.

Next we prove $P^{\oplus P}$ is a lower bound of the square root problem. Note that the complexity class $\oplus P$ is closely related to $\#P$: a set $B$ is in $\oplus P$ iff there exists a function $G$ in $\#P$ such that $B = \{w \in \{0,1\}^* : G(w) = 1 \pmod 2\}$. Thus, we can use below the same construction as in the proof of Theorem 3.3.1.

**Corollary 3.4.3** *For any $A \in \oplus P$, there exist a bounded, simply connected domain $S$ whose boundary $\partial S$ is a Jordan curve represented by a polynomial-time computable function $f : [0,1] \to \mathbb{C}$, and three polynomial-time computable functions $\phi_i : \mathbb{N} \to \mathbb{D}^2$, $i = 1,2,3$, such that for any $n > 0$ and any $w \in \{0,1\}^n$,*

*(a) $\delta(\phi_i(w), \partial S) > 2^{-p(n)}$ for some polynomial function $p$, $i = 1,2,3$.*

*(b) $\phi_1(w) \notin S$ and $\phi_2(w), \phi_3(w) \in S$.*

*(c) $w \in A$ if and only if $(h_S(\phi_3(w), \phi_1(w)) \bmod 2)$ and $(h_S(\phi_2(w), \phi_1(w)) \bmod 2)$ differ by 1.*

*Proof.* Since $A \in \oplus P$, there exists a set $B \in P$ and a polynomial $q$ such that for all $w$,

$$w \in A \Leftrightarrow ||\{u : \ell(u) = q(\ell(w)), \langle w, u\rangle \in B\}|| \text{ is odd.}$$

We define a function $G : \{0,1\}^* \to \mathbb{N}$ as $G(w) = ||\{u : \ell(u) = q(\ell(w)), \langle w, u\rangle \in B\}||$. By Theorem 2.3.2 (c), $G$ is in $\#P$; we further have $w \in A$ iff $G(w)$ is odd. We define the set $S$ and $\phi_i$, for $i = 1,2,3$, exactly as those in the proof of Theorem 3.3.1. Then, they satisfy conditions (a) and (b). Furthermore, condition (c) is also satisfied

since, from the proof of Theorem 3.3.1, we have

$$\begin{aligned} w \in A &\Leftrightarrow G(w) \text{ is odd.} \\ &\Leftrightarrow h_S(\phi_3(w), \phi_1(w)) - h_S(\phi_2(w), \phi_1(w)) \text{ is odd.} \\ &\Leftrightarrow (h_S(\phi_3(w), \phi_1(w)) \bmod 2) \text{ and } (h_S(\phi_2(w), \phi_1(w)) \bmod 2) \text{ differ by 1.} \end{aligned}$$

□

**Corollary 3.4.4** *If $P \neq \oplus P$, then there exists a bounded, simply connected domain $S$ whose boundary $\partial S$ is a polynomial-time computable Jordan curve such that the square root problem on $S$ is not polynomial-time solvable.*

# Chapter 4

# Complexity Theory of $NC$ and Log-space Analytic Functions

## 4.1 Introduction

We are interested in the parallel-time complexity of analytic functions defined on $\mathbb{R}$ or $\mathbb{C}$. The sequential-time complexity of analytic functions has been studied by Ko [40] and Müller [53]. The parallel-time complexity of real functions has been studied by Hoover [36; 37], who introduced the notion of $NC$ computable real functions. $NC$ denotes the class of decision problems solvable by a family of Boolean circuits with polynomial size and polylog depth. In this chapter, by $NC$ we mean *uniform* $NC$, which requires the Boolean circuit family to be constructed by a log-space Turing machine. It is well known that $L \subseteq NC \subseteq P$, where $L$ and $P$ denote the classes of decision problems solvable by log-space Turing machines and polynomial-time Turing machines, respectively. In this chapter, we extend Ko's and Hoover's studies to $NC$ analytic functions.

We first investigate the complexity of derivatives and integration of $NC$ real functions. It is known that the derivatives of polynomial-time computable real func-

tions may have arbitrarily high complexity, and that the complexity of integration of polynomial-time computable real functions may be as high as a $\#P$-complete discrete function (see Ko [40]). We show that these negative results also hold for $NC$ real functions. In fact, we show that the integration of log-space computable real functions may be as hard as the integration of polynomial-time computable real functions, that is, the complexity is still $\#P$.

For analytic functions defined on the complex plane, it is known that if $f$ is a polynomial-time computable, analytic function on a neighborhood containing the origin 0, then the sequence $\{f^{(n)}(0)/n!\}$ is polynomial-time uniformly computable (Ko [40]). We extend this result to $NC$ and log-space analytic functions: If $f$ is an $NC$ or log-space computable, analytic function defined on a neighborhood containing the origin 0, then the sequence $\{f^{(n)}(0)/n!\}$ of derivatives is $NC$ or log-space uniformly computable, respectively. This is a nontrivial extension. We achieve these results through detailed analysis of parallel computation of the derivatives. For example, we use Newton interpolation to approximate the derivatives; for the log-space case, we use the recently proved result that integer division is $NC^1$ computable. As a consequence, the integral $\int f(t)dt$ of an $NC$ or log-space computable analytic function is also $NC$ or log-space computable, respectively.

One of the fundamental algorithmic problems in complex analysis is to find the zeros of an analytic function inside a Jordan curve (see, e.g., Henrici [35]). This problem has been attempted by various methods and algorithms, namely, (1) methods based on the bisection algorithm, which keep searching zeros in subdivided squares using the *principle of the argument* [35] (see, e.g., Yakoubsohn [79] and Meylan et al. [52]); (2) simultaneous iterative methods based on Newton's method, which require to make a good guess at the initial step (see, e.g., Petkovic et al. [60; 61]); and (3) the quadrature methods based on numerical evaluation of integrals, which turn the problem into that of finding all zeros of the associated polynomial function (see,

e.g., Kravanja et al. [48] and Delves et al. [23]). We study this problem from the complexity-theoretic point of view. We study the complexity of finding all zeros of an *NC* analytic function inside a given Jordan curve. We give a careful complexity analysis of the quadrature method and demonstrate that the zeros of an *NC* analytic function $f$ inside an *NC* Jordan curve $\Gamma$ are all *NC* computable if (i) $f$ is analytic on a simply connected domain that covers $\Gamma$, and (ii) there is an absolute constant $c > 0$ such that $|f(\mathbf{z})| > c$ for all $\mathbf{z}$ on or near $\Gamma$.

## 4.2 Complexity of Derivatives and Integration of *NC* Functions

In this section we first study the complexity of computing derivatives and integrals of *NC* real functions, then restrict ourselves to *NC* analytic functions and show how analyticity affects the complexity. These results may be viewed as the *NC* versions of those in Ko [40].

### 4.2.1 Differentiation and Integration of *NC* Real Functions

Before we ask the complexity of finding the derivatives of an *NC* real function, the first question is whether they exist.

**Theorem 4.2.1** *There exists an $NC^1$ function $f : [0,1] \to \mathbb{R}$ such that $f$ is nowhere differentiable.*

*Proof.* The construction is essentially identical to that of Theorem 6.1 of Ko [40]. We omit it here. □

In the following, $C^k[0,1]$ denotes the set of functions $f : [0,1] \to \mathbb{R}$ that has continuous $k$-th derivative $f^{(k)}$ and $C^\infty[0,1]$ denotes the set of infinitely differentiable functions $f : [0,1] \to \mathbb{R}$.

**Theorem 4.2.2** *Let $f : [0,1] \to \mathbb{R}$ be an NC function and have a continuous derivative on $[0,1]$. Then $f'$ is NC computable on $[0,1]$ iff $f'$ has a polynomial modulus of continuity on $[0,1]$. If, furthermore, $f \in C^k[0,1]$ for some $k > 0$, then $f^{(i)}$ is NC computable for $i < k$; if $f \in C^\infty[0,1]$, then $f^{(i)}$ is NC computable for all $i > 0$.*

*Proof.* The proof is essentially identical to that of Theorem 6.2 of Ko [40]. We omit it here. □

**Theorem 4.2.3** *There exists an NC function $f : [0,1] \to \mathbb{R}$ whose derivative $f'$ exists everywhere but $f'(d)$ is not a computable real number for all $d \in \mathbb{D} \cap [0,1]$.*

*Proof.* The construction is essentially identical to that of Theorem 6.4 of Ko [40]. We omit it here. □

Now we consider the complexity of integration. As integration is in some sense a summation, it is not surprising that it is related to counting classes. As pointed out by Ko [40], the complexity of integration of polynomial-time computable real functions is characterized by the counting class $\#P$. Now our question is: what is the complexity of integration of *NC* real functions?

**Theorem 4.2.4** *Let $\mathcal{C}$ be one of complexity classes $\{L, NC, P\}$, and $F\mathcal{C}$ be the corresponding class of functions. The following are equivalent:*

*(a) Let $f : [0,1] \to \mathbb{R}$ be a $\mathcal{C}$ function. Then, $h(x) = \int_0^x f(t)dt$ is a $\mathcal{C}$ function.*

*(b) Let $f : [0,1] \to \mathbb{R}$ be a $\mathcal{C}$ function in $C^\infty[0,1]$. Then, $h(x) = \int_0^x f(t)dt$ is a $\mathcal{C}$ function.*

*(c) $F\mathcal{C} = \#P$.*

*Proof.* We first define a class $\exists\mathcal{C}$ of languages $A$ such that there exist a language $B \in \mathcal{C}$ and a polynomial function $p$ such that for all $w \in \{0,1\}^*$,

$$w \in A \Leftrightarrow (\exists v \in \{0,1\}^{p(\ell(w))})\langle w, v\rangle \in B.$$

Then we define a class $\widetilde{\#}\mathcal{C}$ of functions $g$ such that there exist a language $B \in \mathcal{C}$ and a polynomial function $p$ such that for all $w \in \{0,1\}^*$,

$$g(w) = ||\{u : \ell(u) = p(\ell(w)) \text{ and } \langle w, u\rangle \in B\}||.$$

It is obvious that $\exists P = NP$ and $\widetilde{\#}P = \#P$ (see, e.g., Du and Ko [26]). However, we do not know whether $\exists L = NL$ or $\widetilde{\#}L = \#L$. In fact, it is not hard to prove that $\exists L = \exists NC = NP$ and $\widetilde{\#}L = \widetilde{\#}NC = \#P$, and so it is most likely that $\widetilde{\#}L \neq \#L$ (recall that $\#L \subseteq FNC^2 \subseteq FP$).

The proof of Theorem 5.33 in Ko [40] can now be adapted to prove $(a) \Leftrightarrow (b) \Leftrightarrow F\mathcal{C} = \widetilde{\#}\mathcal{C}$, and the theorem follows the fact that $\widetilde{\#}L = \widetilde{\#}NC = \#P$. □

### 4.2.2 Differentiation and Integration of $NC$ Analytic Functions

In Section 4.2.1, we showed that, for a given $NC$ function $f \in C^\infty[0,1]$, the sequence $\{f^{(n)}\}$ is $NC$ computable, but we remark here that it is not necessarily $NC$ *uniformly* computable (see Bläser [9]). For integration, we have shown that it has higher complexity $\#P$ than $FNC$, with the assumption that $\#P \neq FNC$. In this section, we study the same problems for functions $f$ that are analytic. We consider complex analytic functions instead of real analytic functions, but the results still hold for the real analytic case.

Let $S$ be a domain. A function $f : S \to \mathbb{C}$ is called an *analytic function* if for every point $\mathbf{z} \in S$, $f'(\mathbf{z})$ exists, or equivalently, if there is a power series that converges to $f$ at a neighborhood of $\mathbf{z}$ for every point $\mathbf{z} \in S$. It is obvious that if $f$ is analytic, then it is infinitely differentiable. If an interval $[a,b] \subseteq S$ is on the real line $\mathbb{R}$ and $\{f^{(n)}(a)\}$ are all real numbers, we say $f$ is real analytic on $[a,b]$. Analyticity is a stronger property than continuity and infinite differentiability, and

thus the complexity of operations on analytic functions are sometimes lower than that on more general functions.

We first recall the concepts of *uniform computability* (see Ko [40]). Let $\mathcal{C}$ be one of the complexity classes $\{L, NC, P\}$. By an $L$, $NC$ or $P$ machine, we mean a log-space Turing machine, $NC$ circuit family or polynomial-time Turing machine, respectively.

**Definition 4.2.5** *A sequence $\{x_n\}$ of real (or complex) numbers is $\mathcal{C}$* uniformly computable *if there exists a $\mathcal{C}$ machine $M$ that for all $n, k \geq 0$, $M$ approximates $x_n$ with an error $\leq 2^{-k}$ with the complexity of $\mathcal{C}$, where the complexity is measured in terms of $n$ and $k$. For example, $\{x_n\}$ is $NC$ uniformly computable*[1] *if there exists an $NC^i$ circuit family $\{C_n\}$ for some $i \geq 0$ such that for any $n, k > 0$, $C_{\langle n,k\rangle}$ outputs a dyadic number $d$ such that $|d - x_n| \leq 2^{-k}$. In other words, $\{C_{\langle n,k\rangle}\}_{k=0}^{\infty}$ computes $x_n$. (We also say $\{x_n\}$ is $NC^i$ uniformly computable in order to specify the circuit depth.)*

Similar to the above definition, we can further define $\mathcal{C}$ uniformly computable sequences $\{f_n\}$ of real functions by modifying definitions such as Definition 2.5.3. For example, "$\{f_n\}$ has a uniform polynomial modulus" means that there exists a polynomial function $p$ such that for any $n, k > 0$ and any $x_1, x_2 \in [0, 1]$, $|x_1 - x_2| \leq 2^{-p(n+k)} \Rightarrow |f_n(x_1) - f_n(x_2)| \leq 2^{-k}$.

**Theorem 4.2.6** *Suppose $f$ is an analytic function defined on a domain that contains the closed unit disk, and $\mathcal{C}$ is one of the complexity classes $P$, $L$ or $NC^i$, $i \geq 1$. If $f$ is $\mathcal{C}$ computable, then*

*(a) $\{f^{(n)}(0)/n!\}$ is a $\mathcal{C}$ uniformly computable sequence.*

*(b) $\int_{[0,x]} f(t)dt$ is $\mathcal{C}$ computable.*

[1]The word "uniformly" here refers to the uniform computation of the sequence $\{x_n\}$, and is not to be confused with the uniform computation of the Boolean circuit family in the definition of "uniform $NC$".

*Proof.* Note that the case $\mathcal{C} = P$ has been proved in Ko [40]. We give a sketch of the proof for the case $\mathcal{C} = NC^i$ (and the case $\mathcal{C} = L$ follows from the case $\mathcal{C} = NC^1$ easily). We assume that $f$ is real analytic on $[0, 1]$ in order to simplify the presentation (or we can write $f = f_1 + if_2$ for two real analytic functions $f_1$ and $f_2$).

For $(a)$, let $a_n = f^{(n)}(0)/n!$, $n \in \mathbb{N}$. Assume that we want to compute approximate values of $a_1$, $a_2$, $\cdots$, $a_n$, with error $\leq 2^{-n}$. We use the method of Newton Interpolation. Let $M$ be an upper bound of $|f|$ on the closed unit disk. Let $b = 2^{-cn}$ for some constant $c$ such that $bM(n+1) \leq 2^{-(n+2)}$ and $(1-b)^{n+2} \geq 1/2$. Let $x_k = k \cdot 2^{-(c+1)n}$, $0 \leq k \leq n$, then $0 = x_0 < x_1 < \cdots < x_n \leq b = 2^n x_1$. Let

$$a'_k = \frac{\sum_{j=0}^{k} (-1)^j \binom{k}{j} f(x_{k-j})}{k! x_1^k}, 1 \leq k \leq n.$$

It is known that $a'_k = f^{(k)}(\xi_k)/k!$ for some $\xi_k \in [0, x_k] \subseteq [0, b]$ (see, e.g., Burden et al. [13])[2], so

$$|a'_k - a_k| = \left| \frac{\int_{[0,\xi_k]} f^{(k+1)}(t)dt}{k!} \right| \leq b \max_{t \in [0,b]} |f^{(k+1)}(t)|/k!.$$

Note that $f^{(k+1)}(t) = \frac{(k+1)!}{2\pi i} \int_{|\mathbf{z}-t|=1-b} \frac{f(\mathbf{z})}{(\mathbf{z}-t)^{k+2}} d\mathbf{z}$ (Cauchy's Integral formula [35]). Therefore,

$$\max_{t \in [0,b]} |f^{(k+1)}(t)| \leq \frac{M(k+1)!}{(1-b)^{k+2}},$$

and

$$|a'_k - a_k| \leq \frac{bM(k+1)}{(1-b)^{k+2}} \leq \frac{bM(n+1)}{(1-b)^{n+2}} \leq 2^{-(n+1)}.$$

Next we design a circuit $C_{\langle k,n \rangle}$ of four layers to approximate $a'_k$ with error $\leq 2^{-(n+1)}$.

1. The top layer is a division circuit that approximates $\sum_{j=0}^{k} (-1)^j \binom{k}{j} f(x_{k-j})/(k! x_1^k)$ with error $2^{-(n+2)}$.

[2]This is the only place where we need the assumption of the real analyticity of $f$. Formula $a'_k = f^{(k)}(\xi_k)/k!$ does not hold for the case that $f$ is complex analytic.

2. The second layer has two circuits: one is an addition circuit that computes $\sum_{j=0}^{k}(-1)^j\binom{k}{j}f(x_{k-j})$, the other is a multiplication circuit that computes $k!x_1^k$ (since $x_1^k = 2^{-k(c+1)n}$, this circuit is actually a shift).

3. The third layer contains $k+1$ multiplication circuits to compute $(-1)^j\binom{k}{j}f(x_{k-j})$ for $0 \le j \le k$, and a circuit to compute $k!$.

4. The fourth layer contains $k+1$ circuits to compute $(-1)^j\binom{k}{j}$, and $k+1$ circuits to approximate $f(x_{k-j})$ with error $\le 2^{-(k+k(c+2)n+2)}$, where $0 \le j \le k$.

The first layer and the fourth layer bring an error $\le 2^{-(n+2)}$ each, and so the total error is bounded by $2^{-(n+1)}$. The numbers involved are all of length polynomial in $n$ (for example, $n!$ is represented by a binary string of length $O(n \log n)$). Note that except the one computing $f$, all other circuits involved are in $NC^1$; that is, they are of size polynomial in $n$, and of depth linear in $\log n$. In particular, the circuit to compute $\binom{k}{j}$ is in $NC^1$, since the iterated product of $n$ $n$-bit numbers, as well as the division of two $n$-bit numbers, is computable in $NC^1$ (see Chiu et al. [17]). Thus, the whole circuit is of size polynomial in $n$ and of depth $O(depth(f) + \log n)$, where $depth(f)$ is the depth of the circuit family that computes $f$. This completes the proof of $(a)$.

For $(b)$, we have a proof similar to that in Ko [40] (pages 208–209), namely, we write $f(t) = \Sigma_{n=0}^{\infty} a_n t^n$, and to approximate $\int_{[0,x]} f(t)dt$ with error $\le 2^{-n}$, we only need the first $cn$ terms of the power series, where $c > 0$ is a constant. We can throw other terms out because $|a_k| \le M/r^k$ for some $r > 1$ and all $k$ (which makes the terms $a_k$ very small if $k > cn$), since $f$ is analytic in a domain that contains the closed unit disk. We still do the integration term by term and then sum up, except that we use circuits to compute $a_k$'s and to add the integrals $\int_{[0,x]} a_k t^k dt = a_k x^{k+1}/(k+1)$ up for $0 \le k \le cn$. □

### 4.2.3 Integration of Meromorphic Functions

Now we extend the results on integration of *NC* analytic functions to integration of *meromorphic functions* along Jordan curves.

For a given domain $S$, a meromorphic function $f : S \to \mathbb{C}$ is a function that is analytic in all but possibly a discrete subset of $S$, and at those singularities it must go to infinity like a polynomial (i.e., these exceptional points must be poles and not essential singularities).

First note that, for a meromorphic function $f : S \to \mathbb{C}$ and a Jordan curve $\Gamma \subseteq S$, the integral $\int_\Gamma f$ of $f$ along $\Gamma$ is definable, provided that $f$ has no poles on $\Gamma$. To see this, we observe that $f$ has only a finite number of poles $\mathbf{z}_1, \cdots, \mathbf{z}_m$ $(m \geq 0)$ inside $\Gamma$, and so it can be written as

$$f = \sum_{i=1}^{m} \sum_{j=1}^{n_i} \frac{a_{ij}}{(\mathbf{z} - \mathbf{z}_i)^j} + f_1,$$

where $n_i \in \mathbb{N}$ $(1 \leq i \leq m)$ is the degree of the pole $\mathbf{z}_i$ of $f$, and $f_1$ is an analytic function. Then, for any piecewise linear closed curve $\Gamma' \subseteq S$ such that $f$ has no poles on $\Gamma'$ and $f$ has exactly $m$ poles $\mathbf{z}_1, \cdots, \mathbf{z}_m$ inside $\Gamma'$, $\int_{\Gamma'} f$ is defined and by the residue theory (See, e.g., Henrici [35]), $\int_{\Gamma'} f = 2\pi i \sum_{i=1}^{m} a_{i1}$. Therefore, we can define $\int_\Gamma f = \int_{\Gamma'} f = 2\pi i \sum_{i=1}^{m} a_{i1}$.

We say a Jordan curve $\Gamma$ is *NC* computable if there exists an *NC* function $f : [0, 1] \to \mathbb{C}$ such that $f([0, 1]) = \Gamma$, $f$ is 1-1 on $[0, 1)$ and $f(0) = f(1)$. If $p$ is a modulus function of $f$, we also say it is a modulus function of $\Gamma$.

From the above discussion, and Theorem 4.2.6, we obtain the following result.

**Theorem 4.2.7** *Let $\Gamma$ be an NC computable Jordan curve and $S$ be a simply connected domain that contains $\Gamma$. Let $f$ be a meromorphic function on $S$ that has no poles on $\Gamma$. Assume that $n_0$ is a positive integer such that*

(*a*) *For all $\mathbf{z} \in \Gamma$, $\delta(\mathbf{z}, \partial S) \geq 2^{-n_0}$,*

(*b*) *The function $f$ has no poles in $S_1 = \{\mathbf{z} : \delta(\mathbf{z}, \Gamma) < 2^{-n_0}\}$, and*

(*c*) *$f$ is NC computable in $S_1$.*

*Then $\int_\Gamma f$ is NC computable, with the complexity measured in terms of $2^{p(n_0+1)} + n$, where $p$ is the modulus function of $\Gamma$ and $n$ is the precision parameter. Furthermore, if $\Gamma$ is log-space computable and $f$ is log-space computable in $S_1$, then $\int_\Gamma f$ is log-space computable.*

## 4.3 Finding All Zeros of an Analytic Function Inside a Jordan Curve

We consider in this section the following problem: given a simply connected domain $S$ that contains a Jordan curve $\Gamma$ and an *NC* function $f$ that is analytic in $S$, find all zeros of $f$ inside $\Gamma$.

We assume that the function $f$ has no zeros on $\Gamma$, because it is, in general, undecidable whether a zero of $f$ is on $\Gamma$. This is equivalent to assume that the minimum modulus $\min_{\mathbf{z}\in\Gamma} |f(\mathbf{z})|$ of $f$ on $\Gamma$ is greater than zero. Intuitively, the smaller $\min_{\mathbf{z}\in\Gamma} |f(\mathbf{z})|$ is, the harder it is to compute the zeros of $f$ inside $\Gamma$.

A quadrature method of computing all zeros of an analytic function $f$ inside a Jordan curve $\Gamma$ can be stated as follows.

**(a) Computing the number of zeros.** Compute the number of zeros $n = \frac{1}{2\pi i} \int_\Gamma \frac{f'(\mathbf{z})}{f(\mathbf{z})} d\mathbf{z}$ (by principle of the argument, see, e.g., Henrici [35]).

**(b)Computing Newton sums.** Let $\mathbf{z}_1, \cdots, \mathbf{z}_n$ be all zeros of $f$ inside $\Gamma$. The $p$-th Newton sum $s_p$ is defined as $s_p := \mathbf{z}_1^p + \cdots + \mathbf{z}_n^p$. We can compute $s_p = \frac{1}{2\pi i} \int_\Gamma \mathbf{z}^p \frac{f'(\mathbf{z})}{f(\mathbf{z})} d\mathbf{z}$ (see, e.g., Henrici [35]).

**(c)Computing the associated polynomial.** Compute the associated polynomial $p_n(\mathbf{z})$ for zeros of $f$ in $\Gamma$, where $p_n(\mathbf{z}) := \Pi_{i=1}^{n}(\mathbf{z} - \mathbf{z}_i) =: \mathbf{z}^n + \sigma_1\mathbf{z}^{n-1} + \cdots + \sigma_n$. The coefficients $\sigma_1, \cdots, \sigma_n$ can be computed using Newton's Identities (see, e.g., Carpentier and Dos Santos [15]):

$$\begin{aligned} & s_1 + \sigma_1 = 0 \\ & s_2 + s_1\sigma_1 + 2\sigma_2 = 0 \\ & . \\ & . \\ & . \\ & s_n + s_{n-1}\sigma_1 + \cdots + s_1\sigma_{n-1} + n\sigma_n = 0 \end{aligned} \tag{4.1}$$

**(d) Solving the associated polynomial.** Compute the zeros of $p_n(\mathbf{z})$.

Our main theorem for this problem is as follows:

**Theorem 4.3.1** *Let $S$ be a simply connected domain that contains an NC computable Jordan curve $\Gamma$. Let $f$ be an analytic function in $S$. Assume that there exist two constants $n_0, n_1 \in \mathbb{N}$ such that*

*(a) For all $\mathbf{z} \in \Gamma$, $\delta(\mathbf{z}, \partial S) \geq 2^{-n_0}$,*

*(b) $|f(\mathbf{z})| > 2^{-n_1}$ for all $\mathbf{z} \in S_1 = \{\mathbf{z} \in \mathbb{C} : \delta(\mathbf{z}, \Gamma) \leq 2^{-n_0}\}$, and*

*(c) $f(\mathbf{z})$ is NC computable in $S_1$.*

*Also assume that $f$ has at most $m$ zeros inside $\Gamma$, $m \geq 0$. Then the problem of finding all zeros of $f$ inside $\Gamma$ is NC solvable, with the complexity measured in terms of $2^{p(n_0+1)} + m + n + n_1$, where $n$ is the precision parameter and $p$ is the modulus function of $\Gamma$.*

*Proof.* We check that each of the four steps above is in *NC*. Note that $f'/f$ is *NC* computable with the complexity measured in terms of $n_1 + n$. Then the first two

steps are $NC$ computable by Theorem 4.2.7. Step (c) involves the computation of the inverse of a lower triangular matrix, which is $NC^2$ computable (see, e.g., Bovet et al. [11]). Step (d) was proved to be in $NC$ by Neff [54]. (The known results for steps (c) and (d) need to be adapted to our model. ) □

We remark that when $f$ and $\Gamma$ are log-space computable, so are the first two steps of the above quadrature method. However, we do not know whether step (c) can be done in log-space, and Neff's $NC$ algorithm of computing all zeros of a polynomial is of depth $\log^3 n$. Thus, it is still open whether the problem is log-space solvable for this case.

In addition, the constant $2^{p(n_0+1)}$ in the complexity measure appears large, but it is possible that $m = \Omega(2^{p(n_0+1)})$, which makes $2^{p(n_0+1)}$ less significant in the measure. Moreover, for a fixed function $f$ and a fixed Jordan curve $\Gamma$, $2^{p(n_0+1)}$ is also fixed and the larger $n$ becomes, the less significant $2^{p(n_0+1)}$ is in the measure.

## 4.4 Discussions: Representations of Real Numbers in Parallel Time Complexity

We mentioned in Chapter 2 that there are two representations of real numbers which are used to define the complexity of real numbers, namely, the Cauchy function representation and the general left cut representation. These two representations are equivalent to each other within $P$. However, as to be shown next, they are actually different in $NC$ and $L$ respectively, unless $NC = P$ or $L = P$, respectively.

We consider four representations: Standard left cut, general left cut, binary expansion and Cauchy function representation. More precisely, for the complexity class $NC$, we define the following classes of real numbers.

$NC_{SLC}$**:** the class of real numbers whose standard left cuts are $NC$ computable.

$NC_{BE}$**:** the class of real numbers whose standard Cauchy functions are $NC$ computable.

$NC_{GLC}$**:** the class of real numbers $x$ such that there exists a general left cut of $x$ that is $NC$ computable.

$NC_{CF}$**:** the class of real numbers $x$ such that there exists a Cauchy function of $x$ that is $NC$ computable.

Similarly for $L$, we define $L_{SLC}$, $L_{BE}$, $L_{GLC}$, $L_{CF}$, and for $P$, we define $P_{SLC}$, $P_{BE}$, $P_{GLC}$, $P_{CF}$.

Ko [40] showed that

$$P_{SLC} = P_{BE} \subsetneq P_{GLC} = P_{CF}.$$

Our results show that it is more complicated in the parallel time world.

### 4.4.1 *Absolute* results

We first present some *absolute* results, which do not rely on assumptions on the complexity classes such as $P_1 \neq L_1$. Note that we can obtain trivial results $L_{BE} \subseteq NC_{BE} \subseteq P_{BE}$ and so on, since $L \subseteq NC \subseteq P$. We omit such results.

**Theorem 4.4.1** *$NC_{BE} \subseteq NC_{CF}$, $L_{BE} \subseteq L_{CF}$.*

*Proof.* This is obvious since the standard Cauchy function $b_x$ of a real number $x$ is a Cauchy function of $x$. □

**Theorem 4.4.2** *$NC_{BE} \subseteq NC_{SLC}$, $L_{BE} \subseteq L_{SLC}$.*

*Proof.* For any number $x \in NC_{BE}$, we have that $b_x$ is in $NC$. If $x \in \mathbb{D}$, then $x \in NC_{SLC}$. Assume that $x \notin \mathbb{D}$. For any $n \in \mathbb{N}$ and $d \in \mathbb{D}_n$, it is in $NC$ to compare

$b_x(n)$ and $d$. Because $x \notin \mathbb{D}$, $d \leq b_x(n)$ implies that $d < x$, and $d > b_x(n)$ implies that $d > x$. This completes the proof for $NC_{BE} \subseteq NC_{SLC}$. Similarly, we can prove that $L_{BE} \subseteq L_{SLC}$. □

The following theorem extends Theorem 2.8 of Ko [40].

**Theorem 4.4.3** *There exists a real number $x \in L_{CF} - P_{SLC}$. Thus, $L_{CF} - L_{SLC} \neq \emptyset$ and $NC_{CF} - NC_{SLC} \neq \emptyset$.*

*Proof.* We first define a function $T(n)$ inductively: $T(1) = 1$, and $T(n+1) = 2^{2^{2^{2^{T(n)}}}}$. By a simple diagonalization, we can find a set $A \subseteq \{0\}^*$ such that $A$ is computable in space $T(n)$ but not in space $\log T(n)$ (see, for example, Aho et al. [3]). Without loss of generality, let $0 \in A$. Define

$$x = \sum_{i=1}^{\infty} (2 \cdot \chi_A(0^i) - 1) 2^{-2^{2^{T(i)}}}.$$

First we show that $x \in L_{CF}$. We can compute, for each $n$, a dyadic rational $\phi(n)$ as follows:

(1) find the integer $k$ such that $T(k) \leq \log n < T(k+1)$.

(2) compute $\phi(n) = \sum_{i=1}^{k} (2 \cdot \chi_A(0^i) - 1) 2^{-2^{2^{T(i)}}}$.

It is clear that the value $\phi(n)$ computed above differs from $x$ by at most $2^{-(2^{2^{T(k+1)}}-1)} \leq 2^{-n}$. So the above procedure computes a function $\phi \in CF_x$. Furthermore, since $T(k) \leq \log n$, the computation of $\phi(n)$ can be done in space $O(\log n)$. Thus, $x \in L_{CF}$.

Next we show that $x \notin P_{SLC}$. Assume, by way of contradiction, that $SLC_x$ is computable in polynomial time. We will find a Turing machine $M$ computing $\chi_A(0^n)$ in space $\log T(n)$.

Let $M_a$ be a Turing machine that computes $\chi_A$ in space $T(n)$. The new machine for $\chi_A(0^n)$ works as follows. First, it simulates $M_A$ on inputs 0, $0^2$, $\cdots$, $0^{n-1}$ and

computes $d = \sum_{i=1}^{n-1}(2\chi_A(0^i) - 1)2^{-2^{T(i)}}$. Then it determines whether $d \in SLC_x$ and concludes that $\chi_A(0^n) = 1$ iff $d \in SLC_x$.

It is clear that the above machine $M$ indeed computes $\chi_A$. Assume that $LC_x$ is computable in time $p(n)$ and hence in space $2^{p(n)}$. Then the space usage of the machine $M$ is bounded by $O(\sum_{i=1}^{n-1} T(i)) + 2^{p(2^{T(n-1)})} < 2^{2^{2^{T(n-1)}}} = \log T(n)$ for almost all $n$, which contradicts to the fact that $A$ is not computable in space $\log T(n)$.

In the above we have constructed a number $x \in L_{CF} - P_{SLC}$. As $L_{CF} \subseteq NC_{CF}$ and $L_{SLC} \subseteq NC_{SLC} \subseteq P_{SLC}$, we have $L_{CF} - L_{SLC} \neq \emptyset$ and $NC_{CF} - NC_{SLC} \neq \emptyset$. □

Next we consider general left cuts.

**Theorem 4.4.4** *$NC_{CF} \subseteq NC_{GLC}$, $L_{CF} \subseteq L_{GLC}$.*

*Proof.* For a number $x \in NC_{CF}$, let $\phi \in CF_x$ be in $NC$. Note that the general left cut $L_\phi$ of $x$ associated with $\phi$ is $\{d \in \mathbb{D}_n : n \in \mathbb{N}, d \leq \phi(n)\}$. Similar to the proof of Theorem 4.4.2, it is in $NC$ to compare a dyadic number $d$ and $\phi(n)$. □

### 4.4.2 Results under assumptions $P_1 \neq L_1$ and $P_1 \neq NC_1$

We use a specific coding system for the instantaneous descriptions (IDs) of the computation of a time-bounded Turing machine so that we can discuss the simulation of the machine (see Ko [40]). We assume that our machine $M$ works on two tape symbols: 0 and 1 (and a special blank symbol), has $k$ states $q_1, \cdots, q_k$, uses a single tape, and has a time bound $\psi$. For each string $s \in \{0,1\}^*$ of length $\ell(s) = n$, each ID of the computation of $M(s)$ is encoded by a string in $\{0,1\}^*$ of length $2\psi(n) + 2k + 4$: we encode type symbols 0 and 1 by 01 and 10, respectively, and the blank symbol by 00, and the state symbol $q_i$ by $11(01)^i(10)^{k-i}11$, where the state symbol appears just to the left of the tape symbol that is currently scanned by the tape head. (To see this in another way, at any moment, there are at most $\psi(n)$ symbols (of 0, 1 and

the blank) in the computation, whose length is at most $2\psi(n)$ in our encoding, and the state is of length $2k+4$.) Thus, for any input $s$ of length $n$, the computation of $M(s)$ is encoded by a $(\psi(n)+1)\cdot(2\psi(n)+2k+4)$-bit string $\alpha_0\alpha_1\cdots\alpha_{\psi(n)}$, where $\alpha_i$ is the code of the $i$-th ID in the computation of $M(s)$. Note that we fix the length of the codes for IDs to be exactly $2\psi(n)+2k+4$, and fix the number of IDs in the computation to be exactly $\psi(n)+1$.

We say a function $\phi : \mathbb{N} \to \mathbb{N}$ is *log-space computable* if it is log-space computable when both inputs and outputs are written in the binary form. Note that if $\phi$ is log-space computable, then it is also log-space computable when both inputs and outputs are written in the unary form.

We use the following lemma (Lemma 4.15 of Ko [40]).

**Lemma 4.4.5** *Let $M$ be a Turing machine having $k$ states and having a time bound $\psi$, which is log-space computable. Then for any input $st$ of length $\ell(s) = \ell(t) = \psi(n)+2k+4$ such that $s$ is an ID of $M(u)$ for some string $u$ of length $n$, we can determine, in log space, whether $t$ is the successor of $s$ in the computation of $M(u)$, or $t$ is less than the successor ID, or $t$ is larger than the successor ID. Furthermore, we can compute, in log space, the maximum $m$ such that the first $m$ bits of $t$ agree with those of the successor ID of $s$.*

**Theorem 4.4.6** *If $P_1 \neq L_1$, then $L_{SLC} - L_{CF} \neq \emptyset$.*

*Proof.*

Let $T \in P_1 - L_1$ be computed by a deterministic Turing machine $M$ in time $p(n)$. Assume that $p$ is log-space computable. We use the encoding system described above. Assume that $M$ has $k$ states and so on input $0^n$, an ID of $M(0^n)$ is of length $2\psi(n)+2k+4$. We let $s_{n,i}$, $0 \leq i \leq p(n)$, be the $i$-th ID of the computation of $M(0^n)$. Define a real number $x$ whose binary expansion is

$$x = 0.01\tau(s_{1,0})\tau(s_{1,1})\cdots\tau(s_{1,p(1)})\tau(s_{2,0})\cdots\tau(s_{2,p(2)})\cdots,$$

where $\tau$ is the local translation function defined by $\tau(0) = 01$, $\tau(1) = 10$, and $\tau(ab) = \tau(a)\tau(b)$ for all $a, b \in \{0,1\}^+$.

We first show that $x \in P_{CF} - L_{CF}$. Let $q(n) = 2p(n) + 2k + 4$ and $r(n) = \sum_{i=1}^{n} 2q(i) \cdot (p(i) + 1)$. From $T \in P_1$, it is easy to see that $x \in P_{CF}$. Furthermore, note that for each $0^n$, we need only $\tau(s_{n,p(n)})$, or, from the $(r(n) - 2q(n) + 3)$-rd bit to the $r(n)+2$ bit of the binary expansion of $x$, to determine whether $0^n \in T$. Therefore, if $x \in L_{CF}$, then we can compute, in log space, an approximation value $d$ to $x$ such that $|d - x| \leq 2^{r(n)+4}$, and by our coding system, the first $r(n) + 2$ bits of $d$ must be identical to those of $x$ and so we can determine whether $0^n \in T$ in log space, which contradicts to the assumption of $T \in P_1 - L_1$.

Next we show that $x \in L_{SLC}$, that is, $SLC_x = \{d \in \mathbb{D} : d < x\}$ is in $L$. We only need to consider dyadic numbers of even lengths, because for a dyadic number $d$ of an odd length, we can add a trailing zero to $d$. For a dyadic number $d \in \mathbb{D}_{2n}$, we can tell whether $d < x$ or $d > x$ in log space (note that since $x \notin L_{CF}$, $x \notin \mathbb{D}$ and $d \neq x$). We achieve this by generating the initial IDs $s_{i,0}$ and applying Lemma 4.4.5 successively to each substring $uv$ of $d$, where $u$ is already been verified to be equal to $\tau(s_{i,j})$ for some $j$ and $\ell(v) = \ell(u)$; furthermore, we check whether $v > \tau(w)$, $v = \tau(w)$ or $v < \tau(w)$, where $w$ is the next ID of $u$. If $v > \tau(w)$, $d > x$; if $v < \tau(w)$, $d < x$; if $v = \tau(w)$, we continue with the process. If all bits of $d$ agree with the first $2n$ bits of $x$, $d < x$. □

**Corollary 4.4.7** *If $P_1 \neq NC_1$, then $NC_{SLC} - NC_{CF} \neq \emptyset$.*

*Proof.* The proof is the same as that of Theorem 4.4.6, except that we let $T \in P_1 - NC_1$ and the constructed number $x \in L_{SLC} \subseteq NC_{SLC}$. □

**Corollary 4.4.8** *If $P_1 \neq L_1$, then $L_{BE} \subsetneq L_{SLC}$.*

*Proof.* Because we have $L_{BE} \subseteq L_{CF}$, $L_{BE} \subseteq L_{SLC}$ and $L_{SLC} - L_{CF} \neq \emptyset$. □

**Corollary 4.4.9** *(a) If $P_1 \neq L_1$, then $L_{CF}$ and $L_{SLC}$ are incomparable.*

*(b) If $P_1 \neq NC_1$, then $NC_{CF}$ and $NC_{SLC}$ are incomparable.*

*Proof.* Statement (a) follows Theorems 4.4.6 and 4.4.3. Statement (b) follows Corollary 4.4.7 and Theorem 4.4.3. □

**Corollary 4.4.10** *(a) If $P_1 \neq L_1$, then $L_{CF} \overset{\subset}{\neq} L_{GLC}$.*

*(b) If $P_1 \neq NC_1$, then $NC_{CF} \overset{\subset}{\neq} NC_{GLC}$.*

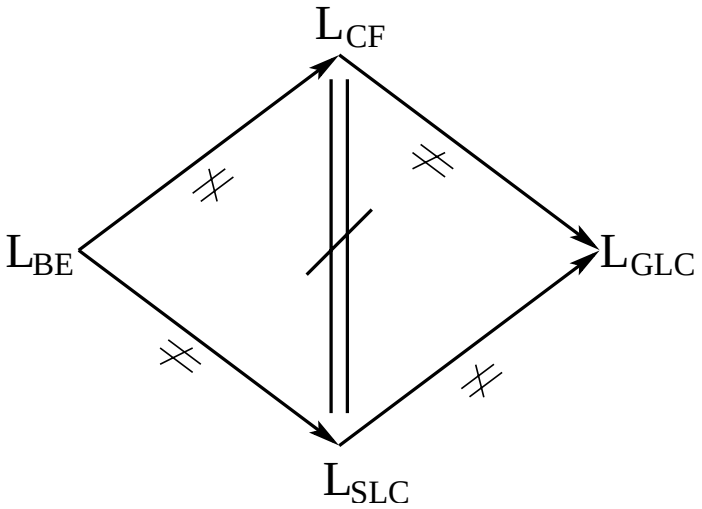


**Figure 4.1:** Assuming $P_1 \neq L_1$.

In summary, if $P_1 \neq L_1$, then $L_{SLC}$, $L_{BE}$, $L_{GLC}$ and $L_{CF}$ are four distinct classes (see Figure 4.1). If $P_1 \neq NC_1$, then $NC_{SLC}$, $NC_{BE}$, $NC_{GLC}$ and $NC_{CF}$ are four distinct classes. The converse also holds, since Ko [40] has shown that $P_{CF} = L_{CF} \Leftrightarrow P_1 = L_1$ and $P_{CF} = NC_{CF} \Leftrightarrow P_1 = NC_1$. We state it as a theorem.

**Theorem 4.4.11** *The following are equivalent:*

*(a)* $P_1 \neq L_1$.

*(b)* $L_{SLC}$, $L_{BE}$, $L_{GLC}$ *and* $L_{CF}$ *are four distinct classes.*

*Similar results hold for NC real numbers.*

The following corollary is an *absolute* result.

**Corollary 4.4.12** $L_{CF} \neq L_{SLC}$, $NC_{CF} \neq NC_{SLC}$.

*Proof.* If $L_{CF} = L_{SLC}$, then from Theorem 4.4.11, $P_1 = L_1$, and from Ko's results, $L_{CF} = P_{CF}$, which implies $L_{SLC} = P_{CF}$ and furthermore $P_{SLC} = P_{CF}$. However, from Theorem 2.8 of Ko [40], $P_{SLC} \subsetneq P_{CF}$. Therefore, $L_{CF} \neq L_{SLC}$. We can prove in a similar way that $NC_{CF} \neq NC_{SLC}$. □

# Chapter 5

# Convex Hulls

## 5.1 Introduction

The convex hull of a set $S$ of the two-dimensional plane is the smallest convex set $CH(S)$ that contains $S$. It is a fundamental concept in mathematics and in computational geometry. For polygons and sets of finite points, there are a number of efficient algorithms to compute their convex hulls (see, for instance, O'Rourke [57] and de Berg et al. [8]). In general, however, no efficient algorithms are known to work for all subsets of the two-dimensional plane. In fact, for some set $S$, its convex hull could be very complicated and defies a simple algorithm.

In this chapter, we study the complexity of computing the convex hull of a given set $S \subseteq \mathbb{R}^2$. In particular, we study two problems about the convex hull $CH(S)$ of a polynomial-time computable set $S \subseteq \mathbb{R}^2$:

> Membership Problem: For a polynomial-time computable set $S$ and a given point $\mathbf{z}$, determine whether $\mathbf{z}$ is inside $CH(S)$.
>
> Area Problem: For a polynomial-time computable set $S$, compute the two-dimensional measure of the convex hull of $S$.

In this chapter, we use the notions of polynomial-time computable Jordan domains, polynomial-time recognizable sets and strongly polynomial-time recognizable sets to study these two problems. (See Section 2.5 for the formal definitions of these notions.) Our main results can be summarized as follows:

(1) There exists a Jordan domain $S \subseteq \mathbb{R}^2$ which is polynomial-time recognizable such that its convex hull is not even recursively recognizable.

(2) If a set $S \subseteq \mathbb{R}^2$ is a Jordan domain and its boundary is polynomial-time computable, or if $S$ is strongly polynomial-time recognizable, then its convex hull $CH(S)$ is strongly nondeterministic polynomial-time recognizable.

(3) If $P \neq NP$, then there exists a Jordan domain $S \subseteq \mathbb{R}^2$ whose boundary is polynomial-time computable such that its convex hull $CH(S)$ is not polynomial-time recognizable.

(4) If a set $S \subseteq \mathbb{R}^2$ is a Jordan domain and its boundary is polynomial-time computable, or if $S$ is strongly polynomial-time recognizable, then the area of its convex hull $CH(S)$ is computable in polynomial-time with the help of an oracle function in $\#P$.

(5) If $FP_1 \neq \#P_1$, then there exists a Jordan domain $S \subseteq \mathbb{R}^2$ whose boundary is polynomial-time computable such that the area of its convex hull $CH(S)$ is not a polynomial-time computable real number.

## 5.2 Convex hull of a $P$-recognizable set

$P$-recognizability is the most general concept of polynomial-time computability for two-dimensional sets, but some of the important properties of a set are not retained in this formulation. For instance, Chou and Ko [20] pointed out that the distance function $\delta_S(\mathbf{z}) = \mathrm{dist}(\mathbf{z}, \partial \mathrm{S})$ is not necessarily computable even if $S$ is $P$-recognizable. It is not hard to see that this is also true for the notion of convex hulls. As a

simple example, suppose $S$ consists of four corners of a square $[0, x] \times [0, x]$ where $x$ is an incomputable real number. Then, $S$ is $P$-recognizable since all its points are on the boundary $\partial S$ and so a trivial oracle TM $M$ that always outputs 0 computes $\chi_S$ correctly for all points away from the boundary. On the other hand, we note that $CH(S)$ is exactly the square $R = [0, x] \times [0, x]$. It is not hard to see that $R$ is recursively recognizable if and only if $x$ is a computable real number.

In the following, we show that, even if $S$ is a Jordan domain and is $P$-recognizable, its convex hull $CH(S)$ is not necessarily recursively recognizable.

**Theorem 5.2.1** *There exists a $P$-recognizable Jordan domain $S$ of which the convex hull $CH(S)$ is not recursively recognizable.*

*Proof.* Let $K \subseteq \mathbb{N}$ be an r.e., nonrecursive set of integers. Then, there exists a TM $M_K$ that enumerates the integers in $K$. That is, $M_K$ prints, on input 0, integers on its output tape one by one such that (i) it prints only integers in $K$, and (ii) every integer in $K$ is eventually printed. For $n \in K$, let $t(n)$ be the number of moves $M_K$ takes to print integer $n$ on input 0. Without loss of generality, we assume that $t(n) \geq 2n + 1$.

Let $O$ denote the origin $\langle 0, 0 \rangle$ of $\mathbb{R}^2$ and $C$ denote the unit circle, i.e., the circle with center $O$ and radius 1. For any $n > 0$, let $a_n = 1/4 - 2^{-(n+1)}$, $Z_n = \langle \cos(2\pi a_n), \sin(2\pi a_n) \rangle$, and $C_n$ be the chord of $C$ connecting the points $Z_n$ and $Z_{n+1}$.

We now define a function $f : [0, 1] \to \mathbb{R}^2$ whose image is a Jordan curve $\Gamma$. On $[1/4, 1]$, the image of $f$ is the circle $C$ on the second, third, and fourth quadrants; i.e., $f(t) = \langle \cos(2t\pi), \sin(2t\pi) \rangle$, for $t \in [1/4, 1]$. Next, for each $n > 0$, if $n \notin K$, then $f$ is linear on $[a_n, a_{n+1}]$, with $f(a_n) = Z_n$ and $f(a_{n+1}) = Z_{n+1}$; i.e., $f$ maps $[a_n, a_{n+1}]$ linearly to the chord $C_n$. If $n > 0$ and $n \in K$, then $f$ maps $[a_n, a_{n+1}]$ to the chord $C_n$ with a bump in the middle, where the bump has width $2^{-t(n)}$ and height $h_n = 1 - \cos(2^{-(n+2)}\pi)$. To be more precise, let $X'_n$ be the middle point of the chord $C_n$, and $X_n$ the intersection point of the circle $C$ and the half line

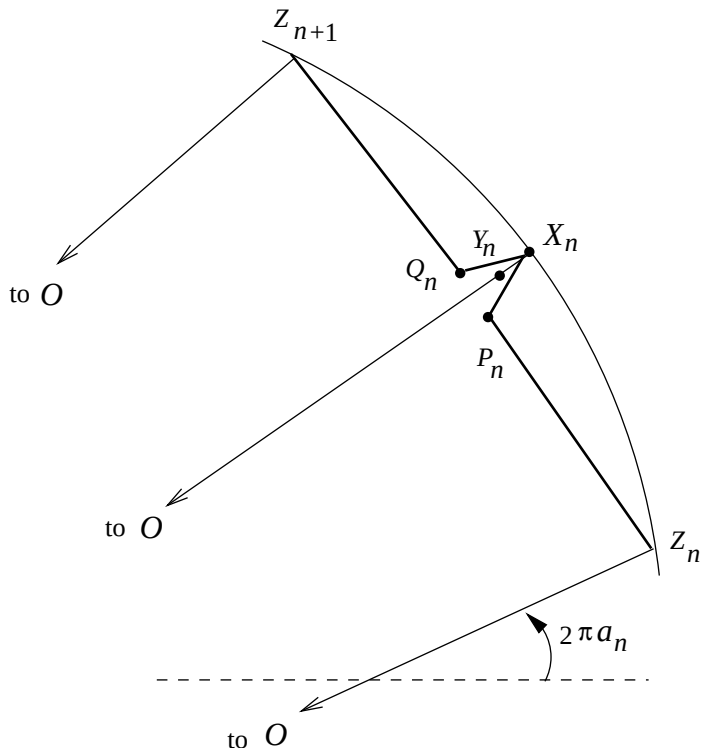


**Figure 5.1:** The curve $\partial S$ between $Z_n$ and $Z_{n+1}$

$\overrightarrow{OX}$. Define $P_n$ and $Q_n$ to be the two points on $C_n$ with distance $2^{-t(n)-1}$ from $X'_n$ (with $P_n$ closer to $Z_n$ and $Q_n$ closer to $Z_{n+1}$).[1] The function $f$ is piecewise linear on $[a_n, a_{n+1}]$ with $f(a_n) = Z_n$, $f((a_n+a_{n+1})/2-2^{-t(n)-n-3}) = P_n$, $f((a_n+a_{n+1})/2 = X_n$, $f((a_n+a_{n+1})/2+2^{-t(n)-n-3}) = Q_n$, and $f(a_{n+1}) = Z_{n+1}$. (Figure 5.1 shows the curve $\Gamma$ between $Z_n$ and $Z_{n+1}$.) This completes the definition of function $f$. Note that $f$ is a continuous function but is not computable.

Let $S$ be the interior of the Jordan curve $\Gamma$. We claim that $S$ is $P$-recognizable. First, it is easy to see that the set $S_0$ that is enclosed by the curve $f[1/4, 1]$ plus all chords $C_n$, for $n > 0$, is $P$-recognizable. Next let $B_k$ be the area enclosed by the chord $C_n$ and the circle $C$ from $Z_n$ to $Z_{n+1}$, and let $S_k = S \cap B_k$. If $k \notin K$, then $S_k = \emptyset$; and if $k \in K$, then $S_k$ is a small bump of width $2^{-t(n)}$ and height $h_n$. Now,

[1]Note that $t(n) \geq 2n+1$ implies that $2^{-t(n)-1} \leq 2^{-2n-2}$, and the distance between $Z_n$ and $X'_n$ is $\sin(2^{-n-2}\pi) > 2^{-n-2}$. Therefore, $P_n$ and $Q_n$ are between $Z_n$ and $Z_{n+1}$.

consider the following algorithm for the membership problem of $S$:

**Oracles**: $\langle \phi_1, \phi_2 \rangle$ representing a point $\mathbf{z} \in \mathbb{R}^2$.

**Input**: $n > 0$.

(1) Ask the oracles to get a dyadic point $\mathbf{w} \in \mathbb{D}^2_{n+1}$ with $|\mathbf{w} - \mathbf{z}| < 2^{-(n+1)}$.

(2) If $\mathbf{w} \in S_0$, then answer YES;

(3) Else if $\mathbf{w} \notin B_k$ for any $k \leq n$, then answer NO;

(4) Else if $\mathbf{w} \in B_k$ for some $k \leq n$, then simulate TM $M_K$ for $n$ moves, and answer YES if and only if $M_K$ prints $k$ within $n$ moves and $\mathbf{w} \in S_k$.

To see that the above algorithm solves the membership problem of $S$ correctly, assume that $\mathbf{z}$ is a point in $\mathbb{R}^2$ with $\mathrm{dist}(\mathbf{z}, \Gamma) > 2^{-\mathrm{n}}$. Then, if $\mathbf{z} \in S_0$ or if $\mathbf{z}$ lies outside $C$, then the answer given by the algorithm is correct. Next, suppose $\mathbf{z} \in B_k$ for some $k > 0$. If $k \notin K$, or if $k \in K$ and $t(k) \leq n$, then again the answer is correct. Finally, suppose $\mathbf{z} \in B_k$ with $k \in K$ and $t(k) > n$. Then, $S_k$ is a small bump of width $2^{-t(k)} < 2^{-n}$, and so all points in $S_k$ have distance at most $2^{-(n+1)}$ from the boundary $\Gamma$. Thus, the answer NO is correct for $\mathbf{z}$ if it has distance $> 2^{-n}$ from $\Gamma$.

Next, we verify that this algorithm works in polynomial time. It is apparent that steps (1)—(3) and the first half of step (4) can be done in time polynomial in $n$. For the second half of step (4), we note that if $t(k) > n$, then we can simulate $M_k$ for $n$ steps and answer NO. Otherwise, if $t(k) \leq n$, then we can calculate $t(k)$ in $O(n)$ moves, and compute points $X_n, P_n, Q_n$ correctly within error $2^{-(n+1)}$ in time polynomial in $n$. From these points, we can then determine whether $\mathbf{w} \in S_k$ if $\mathbf{w}$ has distance $> 2^{-(n+1)}$ from the line segments $\overline{P_n X_n}$, $\overline{X_n Q_n}$. This completes the proof that $S$ is $P$-recognizable.

Now, let us consider the convex hull $CH(S)$ of set $S$. For each $n > 0$, let $T_n = CH(S) \cap B_n$. Note that the curve $\Gamma$ lies completely within $C$ and it includes all

points $Z_n$. Therefore, $T_n$ depends only on the curve $\Gamma$ between $Z_n$ and $Z_{n+1}$. That is, for $n \notin K$, $T_n = \emptyset$; and for $n \in K$, $T_n$ is equal to $\Delta Z_n X_n Z_{n+1}$, the triangle with the vertices $Z_n$, $X_n$ and $Z_{n+1}$. Now, suppose that $CH(S)$ is recursively recognizable. Then, we can determine whether $n \in K$ as follows:

> Let $Y_n$ be the middle point in $\overline{X'_n X_n}$, and determine whether $Y_n$ is inside $CH(S)$ with error $\leq 2^{-2n-6}$. Answer $n \in K$ if and only if $Y_n \in CH(S)$.

Note that $h_n = 1 - \cos(2^{-(n+2)}\pi) \geq 2^{-2n-4}$, and the length of $C_n$ is $2\sin(2^{-(n+2)}\pi) \geq 2^{-n-2}$.[2] Now, it is not hard to see that the distance between $Y_n$ and the boundary of $CH(S)$ is greater than $h_n/4$, no matter whether $n \in K$ (or, equivalently, whether $Y_n \in CH(S)$). Thus, the above algorithm determines whether $n \in K$ correctly. This is a contradiction to the assumption that $K$ is not a recursive set. □

## 5.3 Convex hull of a $P$-computable Jordan domain

In this section, we consider the complexity of convex hulls of $P$-computable Jordan domains. In order to characterize the complexity of convex hulls, we need to extend the notion of $P$-recognizable sets to $NP$-recognizable sets.

**Definition 5.3.1** *(a) A set $T \subseteq \mathbb{R}^2$ is NP-*recognizable *if there exists a polynomial-time nondeterministic oracle TM $M$ such that, on oracles $\langle \phi_1, \phi_2 \rangle$ representing a point $\mathbf{z} \in \mathbb{R}^2$, and on input $n > 0$,*

*(i) For $\mathbf{z} \in T$ with $\mathrm{dist}(\mathbf{z}, \partial \mathrm{T}) > 2^{-\mathrm{n}}$, $M^{\phi_1,\phi_2}(n)$ contains at least one accepting path, and*

*(ii) For $\mathbf{z} \notin T$ with $\mathrm{dist}(\mathbf{z}, \partial \mathrm{T}) > 2^{-\mathrm{n}}$, $M^{\phi_1,\phi_2}(n)$ has no accepting paths.*

[2] By the Taylor expansion of the functions cos and sin, we know that for small $t$, $1 - \cos t \geq t^2/2 - t^4/24 \geq t^2/4$, and $2\sin t \geq 2(t - t^3/6) \geq t$.

*(b) A set $T \subseteq \mathbb{R}^2$ is* strongly *NP*-recognizable *if it is NP-recognizable and the nondeterministic oracle TM $M$ also satisfies the following stronger condition*

*(i′) For all $\mathbf{z} \in T$, $M^{\phi_1,\phi_2}(n)$ contains at least one accepting path.*

**Theorem 5.3.2** *Assume that $S \subseteq [0,1]^2$ is a Jordan domain whose boundary $\partial S$ is P-computable. Then, its convex hull $CH(S)$ is strongly NP-recognizable.*

*Proof.* Let $S^{c\ell}$ denote the closure of $S$; i.e., $S^{c\ell} = S \cup \partial S$. We note that, as $S$ is a Jordan domain, $CH(S^{c\ell}) = CH(S) \cup CH(S)^{c\ell}$. Since the notion of $P$- and $NP$-recognizable sets allows the machine to have errors near the boundary of the set, $CH(S)$ and $CH(S^{c\ell})$ have the same complexity as far as we are only concerned with these complexity notions. So, in the following, we will work directly with the convex hull $CH(S^{c\ell})$ of the closed set $S^{c\ell}$.

We note that a point $\mathbf{z}$ belongs to $CH(S^{c\ell})$ if and only if there exist three points on the boundary $\partial S$ such that $\mathbf{z}$ lies in the triangle $D$ formed by these three points. The following algorithm for the membership problem of $CH(S)$ is based on this idea.

Assume that the function $f : [0,1] \to \mathbb{R}^2$ represents the boundary $\partial S$, and that $f$ is computable in time $p(n)$ for some polynomial $p$.

**Oracles**: $\langle \phi_1, \phi_2 \rangle$ representing a point $\mathbf{z} \in \mathbb{R}^2$.

**Input**: $n > 0$.

(1) Ask oracles $\langle \phi_1, \phi_2 \rangle$ to get a dyadic point $\mathbf{w} \in {\mathbb{D}_{n+3}}^2$ such that $|\mathbf{w} - \mathbf{z}| \leq 2^{-(n+2)}$.

(2) Nondeterministically guess three dyadic numbers $d_1, d_2, d_3 \in \mathbb{D}_{p(n+3)}$.

(3) Compute three dyadic points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in {\mathbb{D}_{n+4}}^2$ such that $|\mathbf{x}_i - f(d_i)| \leq 2^{-(n+3)}$ for $i = 1, 2, 3$.

(4) Let $D$ be the triangle whose three vertices are $\mathbf{x}_1, \mathbf{x}_2$ and $\mathbf{x}_3$. Accept $\mathbf{z}$ if $\mathbf{w}$ is inside $D$ or has distance $\leq 2^{-(n+1)}$ from the boundary $\partial D$ of $D$.

It is clear that the above algorithm works in polynomial time. To see that the above algorithm strongly recognizes $CH(S^{c\ell})$, we first assume that $\mathbf{z} \in CH(S^{c\ell})$. Then, there must be three numbers $t_1, t_2, t_3 \in [0,1]$ such that $\mathbf{z}$ lies in the triangle $D_0$ formed by three vertices $f(t_1), f(t_2)$ and $f(t_3)$.

Suppose that, for each $i = 1, 2, 3$, we have a dyadic number $d_i \in \mathbb{D}_{p(n+4)}$ and dyadic point $\mathbf{x}_i \in {\mathbb{D}_{n+4}}^2$ such that $|d_i - t_i| \leq 2^{-p(n+3)}$, and $|\mathbf{x}_i - f(d_i)| \leq 2^{-(n+3)}$. Then, $|\mathbf{x}_i - f(t_i)| \leq 2^{-(n+2)}$. Let $D$ be the triangle with $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ as the three vertices. Then, the Hausdorff distance between $D_0$ and $D$ is $\leq 2^{-(n+2)}$. Therefore, $\mathbf{z}$ either lies inside $D$ or has distance $\leq 2^{-(n+2)}$ from $\partial Q$. It follows that $\mathbf{w}$ either lies inside $D$ or has distance $\leq 2^{-(n+1)}$ from $\partial Q$. Therefore, the computation path of the algorithm that guesses the numbers $d_1, d_2, d_3$ will accept $\mathbf{z}$.

Conversely, assume that the above algorithm accepts $\mathbf{z}$, with the guesses $d_1, d_2, d_3 \in \mathbb{D}_{p(n+3)}$. Then, the algorithm found a triangle $D$ such that $\mathbf{w}$ is either inside $D$ or has distance $\leq 2^{-(n+1)}$ from $\partial D$. Let $D_1$ be the triangle with the three vertices $f(d_1), f(d_2)$ and $f(d_3)$. Then, the Hausdorff distance between $D$ and $D_1$ is $\leq 2^{-(n+3)}$. It follows that $\mathbf{w}$ is either inside $D_1$ or within distance $2^{-(n+1)} + 2^{-(n+3)}$ from $\partial D_1$. Since $|\mathbf{w} - \mathbf{z}| \leq 2^{-(n+2)}$, and since $D_1 \subseteq CH(S^{c\ell})$, the point $\mathbf{z}$ is either inside $CH(S^{c\ell})$ or within distance $2^{-n}$ from the boundary of $CH(S^{c\ell})$. This shows that the acceptance of the algorithm is correct. □

**Corollary 5.3.3** *Assume that $S \subseteq [0,1]^2$ is strongly $P$-recognizable. Then, its convex hull $CH(S)$ is strongly $NP$-recognizable.*

*Proof.* Assume that an oracle TM $M$ strongly $P$-recognizes $S$ in time $p(n)$. We modify the algorithm of Theorem 5.3.2 by replacing steps (2) and (3) with

(2′) Guess three points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathbb{D}^2_{n+3}$, and verify that $M^{\mathbf{x}_i}(n+3) = 1$ for $i = 1, 2, 3$,

where $M^{\mathbf{x}_i}(n)$ denotes the computation of $M$ on input $n$ with the standard Cauchy functions of $\mathbf{x}_i$ as the oracles. Then, this new nondeterministic oracle TM strongly accepts $CH(S^c)$. □

Next, we show that the result of strong $NP$-recognizability of the convex hulls cannot be improved to $P$-recognizability, unless $P = NP$.

**Lemma 5.3.4** *For any set $A \in NP$, there exist a $P$-computable Jordan domain $S$, a $P$-computable (discrete) function $g : \{0,1\}^* \to \mathbb{D}$, and a polynomial function $q$, such that, for any $w \in \{0,1\}^*$,*

*(i) The distance between $g(w)$ and the boundary of $CH(S)$ is at least $2^{-q(\ell(w))}$, and*

*(ii) $w \in A$ if and only if $g(w) \in CH(S)$.*

*Proof.* Let $A \in NP$. Then there exist a polynomial function $p$ and a set $B \in P$ such that, for all $w \in \{0,1\}^*$,

$$w \in A \Longleftrightarrow (\exists u, \ell(u) = p(\ell(w)))\langle w, u\rangle \in B.$$

For any $w \in \{0,1\}^*$, we let $i_w$ be the integer between 0 and $2^{\ell(w)} - 1$ whose $\ell(w)$-bit binary representation (with possible leading zeros) is equal to $w$. Also let $w'$ denote the successor of $w$ in the lexicographic ordering. Now, suppose $\ell(w) = n > 0$, we define a dyadic rational number in $[0, 1/4]$: $x_w = (1 - 2^{-(n-1)} + i_w \cdot 2^{-2n})/4$, and an interval: $I_w = [x_w, x_{w'}]$. Note that $I_w$ has length $2^{-2\ell(w)-2}$.

Next, for each $u \in \{0,1\}^{p(n)}$, we define two dyadic rationals and two subintervals of $I_w$ as follows:

$$\begin{aligned}
x_{w,u} &= x_w + 2^{-2n-4} + i_u \cdot 2^{-p(n)-2n-4}, \\
x'_{w,u} &= x_w + 2^{-2n-3} + i_u \cdot 2^{-p(n)-2n-4} = x_{w,u} + 2^{-2n-4}, \\
I_{w,u} &= [x_{w,u}, x_{w,u} + 2^{-p(n)-2n-4}], \\
I'_{w,u} &= [x'_{w,u}, x'_{w,u} + 2^{-p(n)-2n-4}].
\end{aligned}$$

Now, we describe the boundary $\partial S$ of the desired Jordan domain $S$. Let $O$ be the origin, and $C$ the unit circle with center $O$ and radius 1. For each $w \in \{0,1\}^*$ of length $n$, let $Z_w = \langle \cos(2\pi x_w), \sin(2\pi x_w) \rangle$, and $C_w$ the chord connecting $Z_w$ and $Z_{w'}$. Then, length of $C_w$ is equal to $2\sin(2^{-2n-2}\pi)$. We denote it by $leng(C_w)$. Let $X_w$ be the middle point on the arc of $C$ between $Z_w$ and $Z_{w'}$, and $h_n$ be the distance between $X_w$ and the chord $C_w$; that is, $h_n = 1 - \cos(2^{-2n-2}\pi)$. Let $B_w$ denote the area between the chord $C_w$ and the arc of $C$ from $Z_w$ through $X_w$ to $Z_{w'}$.

We now divide each chord $C_w$ into four line segments of equal length, and further divide each of the two middle segments into $2^{p(n)}$ subsegments, each corresponding to a string $u \in \{0,1\}^{p(n)}$. That is, let $V_w$, $V'_w$, and $V''_w$ be the points on $C_w$ of distance $(1/4)leng(C_w)$, $(1/2)leng(C_w)$, and $(3/4)leng(C_w)$ from $Z_w$, respectively. Also let $P_{w,u}$ be the point on $C_w$ of distance $(i_u \cdot 2^{-p(n)-2} \cdot leng(C_w))$ from $V_w$, and $P'_{w,u}$ the point on $C_w$ of distance $(i_u \cdot 2^{-p(n)-2} \cdot leng(C_w))$ from $V'_w$. Finally, let $Q_{w,u}$ be the point in $B_w$ that is of equal distance from $P_{w,u}$ and $P_{w,u'}$ and has distance $h_n/2$ from the chord $C_w$, and $Q'_{w,u}$ the point in $B_w$ that is of equal distance from $P'_{w,u}$ and $P'_{w,u'}$ and has distance $h_n/2$ from the chord $C_w$ (see Figure 5.2).

Now, we are ready to define the function $f : [0,1] \to \mathbb{R}^2$ that computes the boundary $\partial S$ of the desired Jordan domain $S$. First, $f$ maps $[1/4, 1]$ to the unit circle $C$ on the second, third, and fourth quadrants; i.e., $f(t) = \langle \cos(2t\pi), \sin(2t\pi) \rangle$, for $t \in [1/4, 1]$. Next, on each interval $I_w = [x_w, x_{w'}]$, $f$ maps $[x_w, x_w + 2^{-2n-4}]$ linearly to the line segment $\overline{Z_w V_w}$, and maps $[x_w + 3 \cdot 2^{-2n-4}, x_{w'}]$ linearly to the line segment $\overline{V''_w Z_{w'}}$. For each $u \in \{0,1\}^{p(n)}$, if $\langle w, u \rangle \notin B$, then $f$ maps $I_{w,u}$ linearly to the line segment $\overline{P_{w,u} P_{w,u'}}$, and maps $I'_{w,u}$ linearly to the line segment $\overline{P'_{w,u} P'_{w,u'}}$. If $\langle w, u \rangle \in B$, then $f$ maps $I_{w,u}$ piecewise linearly to two line segments: $\overline{P_{w,u} Q_{w,u}}$ and $\overline{Q_{w,u} P_{w,u'}}$, and maps $I'_{w,u}$ piecewise linearly to two line segments $\overline{P'_{w,u} Q'_{w,u}}$ and $\overline{Q'_{w,u} P'_{w,u'}}$. This completes the definition of $f$. Finally, we let $g(w)$ be the point $Y_w$ between $O$ and $X_w$ that has distance $3 \cdot h_n/4$ from $X_w$. Figure 5.2 shows the curve $\partial S$ in the area

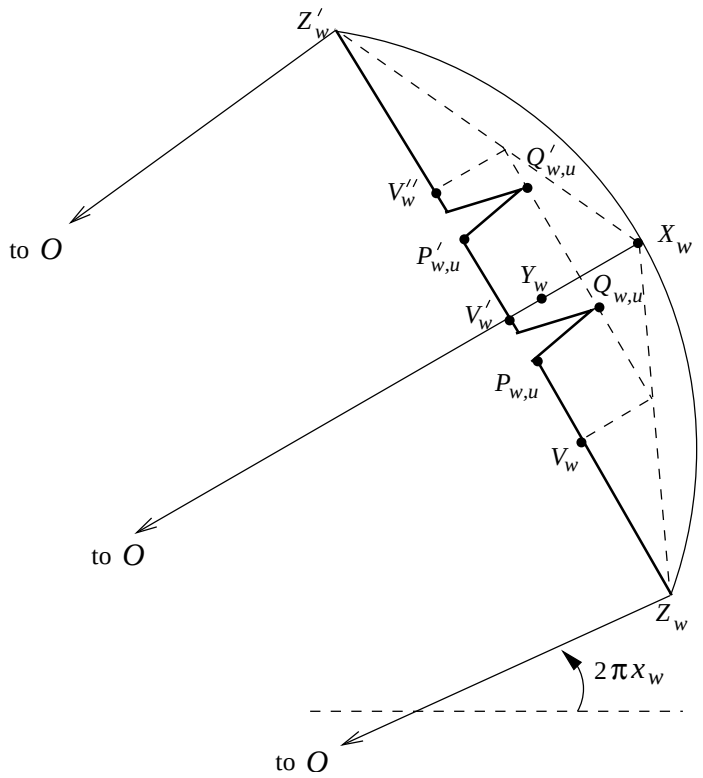


**Figure 5.2:** The curve $\partial S$ around $C_w$

$B_w$.

It is not hard to see that the function $f$ and $g$ are polynomial-time computable. We omit the details of the proof.

We now check that the domain $S$ satisfies the required conditions. As we argued in the proof of Theorem 5.2.1, our design of the curve $\partial S$ guarantees that the part of the convex hull $CH(S)$ within $B_w$ depends only on the curve $\partial S$ between $Z_w$ and $Z_{w'}$. More precisely, if $w \notin A$, then $CH(S) \cap B_w = \emptyset$, and $Y_w \notin CH(S)$. On the other hand, if $w \in A$, then $S \cap B_w$ contains at least two bumps which lie to the two sides of $Y_w$, and so $Y_w \in CH(S)$. Furthermore, we claim that, no matter whether $Y_w \in CH(S)$, the distance between $Y_w$ and the boundary of $CH(S)$ is greater than $2^{-p(n)-4n-5}$.

For the case of $Y_w \notin CH(S)$, we know that the chord $C_w$ is part of the boundary of $CH(S)$, and $\mathrm{dist}(\mathrm{Y_w}, \mathrm{C_w}) = \mathrm{h_n}/4$. For the case of $Y_w \in CH(S)$, let us assume

that $\partial S$ passes through two points $Q_{w,u}$ and $Q'_{w,u}$. Then, the line segment $\overline{Q_{w,u}Q'_{w,u}}$ forms part of the boundary of the convex hull $CH(S)$, and $Y_w$ has distance $h_n/4$ from this boundary. In addition, we know that both $Q_{w,u}$ and $Q'_{w,u}$ have distance at least $(2^{-p(n)-3} \cdot leng(C_w))$ away from the line $\overline{OX_w}$. It implies that $Y_w$ has distance at least $(2^{-p(n)-3} \cdot leng(C_w))$ from other parts of the boundary of $CH(S)$. That is, no matter whether $Y_w \in CH(S)$, $\mathrm{dist(Y_w, \partial S)} \geq \min\{\mathrm{h_n}/4, 2^{-\mathrm{p(n)}-3} \cdot \mathrm{leng(C_w)}\}$.

Note that $h_n = 1 - \cos(2^{-2n-2}\pi) \geq 2^{-4n-3}$, and $leng(C_w) = 2\sin(2^{-2n-2}\pi) \geq 2^{-2n-2}$. Therefore, $\mathrm{dist(Y_w, \partial S)} \geq 2^{-\mathrm{p(n)}-4\mathrm{n}-5}$. This completes the proof of the claim. The proof of the lemma is also complete by setting $q(n) = p(n) + 4n + 5$. □

**Theorem 5.3.5** *Assume that $P \neq NP$. Then, there exists a Jordan domain $S \subseteq \mathbb{R}^2$ whose boundary $\partial S$ is $P$-computable but whose convex hull $CH(S)$ is not $P$-recognizable.*

*Proof.* Assume that the convex hull $CH(S)$ of the set $S$ constructed in Lemma 5.3.4 is $P$-recognizable. Then, we can determine whether $w \in A$ by asking whether $g(w)$ is in $CH(S)$, with error bound $< 2^{-q(n)}$. □

**Corollary 5.3.6** *Assume that $P \neq NP$. Then, there exists a Jordan domain $S \subseteq \mathbb{R}^2$ which is strongly $P$-recognizable but whose convex hull $CH(S)$ is not $P$-recognizable.*

## 5.4 Areas of Convex Hulls

In this section, we consider the complexity of computing the area of the convex hull $CH(S)$ of a $P$-computable Jordan domain $S$. We first recall the results about the complexity of computing the area of a set $T$ in the two-dimensional plane.

**Proposition 5.4.1** *(a) If $T \subseteq [0,1]^2$ is $P$-approximable, then area of $T$ is a real number in $\#P_{\mathbb{R}}$.*

*(b) If $T \subseteq [0,1]^2$ is a $P$-recognizable Jordan domain with a rectifiable boundary, then area of $T$ is in $\#P_{\mathbb{R}}$.*

*(c) If $FP_1 \neq \#P_1$, then there exists a convex set $T \subseteq [0,1]^2$ that is $P$-approximable but its area is not in $P_{\mathbb{R}}$.*

*Remarks.* (1) Friedman [29] proved that the integral $\int_0^1 f$ of a $P$-computable function $f : [0,1] \to \mathbb{R}$ is a real number in $\#P_{\mathbb{R}}$. Parts (a) and (b) of Proposition 5.4.1 are due to Chou and Ko [18], in which the result of Friedman [29] was extended to the measure of two-dimensional $P$-approximable and $P$-recognizable sets.

(2) Friedman [29] also showed that, if $FP \neq \#P$, then the integral $\int_0^1 f$ of some $P$-computable function $f : [0,1] \to \mathbb{R}$ is not in $P_{\mathbb{R}}$. Du and Ko [25] and Chou and Ko [18] extended this result to two-dimensional, $P$-approximable, convex sets.

We note that a convex Jordan domain $T$ must have a rectifiable boundary. Therefore, if the convex hull $CH(S)$ of a Jordan domain is $P$-recognizable, then its area is a real number in $\#P_{\mathbb{R}}$. This observation can be easily extended to $NP$-recognizable convex hulls.

We let $\#NP_{\mathbb{R}}$ denote the class of real numbers $x$ which have a Cauchy function representation $\phi : \{0\}^* \to \mathbb{D}$ such that the function $\phi'(0^n) = \phi(n) \cdot 2^n$ is a function in $\#NP$.

**Theorem 5.4.2** *Assume that $S$ is a $P$-computable Jordan domain. Then, the area of $CH(S)$ is a real number in $\#NP_{\mathbb{R}}$.*

*Proof.* Without loss of generality, assume that $S \subseteq [0,1]^2$. Also assume that the boundary of $CH(S)$ has length bounded by $a$. Assume that $M$ is a nondeterministic polynomial-time oracle TM that strongly $NP$-recognizes $CH(S)$, as given in Theorem 5.3.2. For any $n > 0$, let

$$B = \{\langle 0^n, d_1, d_2\rangle \mid d_1, d_2, \in \mathbb{D}_n, M^{d_1,d_2}(n) \text{ accepts}\},$$

where $M^{d_1,d_2}$ denotes the computation of the machine $M$ using the standard Cauchy functions for $d_1$ and $d_2$ as the oracles. It is clear that $B \in NP$. Furthermore, the function

$$\phi(0^n) = \|\{\langle d_1, d_2\rangle \mid d_1, d_2 \in \mathbb{D}_n, \langle 0^n, d_1, d_2\rangle \in B\}\|$$

is a function in $\#NP$ such that the function $\psi(0^n) = \phi(0^n) \cdot 2^{-2n}$ converges to the area of $CH(S)$ with error $|\psi(0^n) - area(CH(S))| \leq a \cdot 2^{-2n+2}$. □

Next, we study whether $CH(S)$ is actually a real number in $\#P_{\mathbb{R}}$. For this question, we need to review more results about the relations between counting complexity classes in discrete complexity theory.

In his celebrated paper about counting complexity classes, Toda [70] showed that $PP^{PH} \subseteq P^{\#P[1]}$; that is, if a set is computable in probabilistic polynomial time relative to a set in the polynomial-time hierarchy, then it is computable in polynomial-time with a single query to an oracle function in $\#P$.[3] Toda and Watanabe [71] further extended this result to the function classes and showed that $\#P^{PH} \subseteq FP^{\#P[1]}$. Since $\#NP$ is a subclass of $\#P^{PH}$, the following result is immediate.

**Proposition 5.4.3** $\#NP \subseteq FP^{\#P[1]}$.

Combining Propositions 5.4.1 and 5.4.3, we obtain the following results about the area of $CH(S)$.

**Corollary 5.4.4** *Assume that $S \subseteq \mathbb{R}^2$ is a P-computable Jordan domain. Then, the area of $CH(S)$ is a real number in $P_{\mathbb{R}}^{\#P}$.*

**Corollary 5.4.5** *The following are equivalent:*

*(a) For any P-computable Jordan domain $S \subseteq \mathbb{R}^2$, the area of $CH(S)$ is in $P_{\mathbb{R}}$.*

*(b) $FP_1 = \#P_1$.*

[3]Here, $PP$ denotes the class of sets accepted by polynomial-time probabilistic TMs with accepting probability greater than 1/2. For more details, see Du and Ko [26].

Corollary 5.4.4 leaves it open whether the area of $CH(S)$ is actually in $\#P_{\mathbb{R}}$. This question is clearly related to the question of whether the discrete classes $\#P$ and $\#NP$ are equal. The following corollary follows Theorem 2.3.2 (f), which states $NP = UP$ if and only if $\#P = \#NP$.

**Corollary 5.4.6** *If $UP = NP$, then area of the convex hull $CH(S)$ of a P-computable Jordan domain $S$ is in $\#P_{\mathbb{R}}$.*

Whether the converse of the above holds remains open. We note that Theorem 2.3.2(f) implies that if $UP \neq NP$ then there exists some function $\psi$ in $\#NP$ that is not in $\#P$. However, this function $\psi$ constructed in the proof in Hemaspaandra and Vollmer [34] is a simple, characteristic function of a set $A \in NP - UP$. It seems difficult to construct a P-computable Jordan curve $S$ of which the area of $CH(S)$ is related to such a function $\psi$. It would be interesting to find out whether a stronger condition of separating some discrete classes implies that the area of $CH(S)$ is not in $\#P_{\mathbb{R}}$.

# Chapter 6

# Circumscribed Rectangles and Squares for a Two-Dimensional Domain

## 6.1 Introduction

Let $S \subseteq \mathbb{R}^2$ be a bounded, connected domain in the two-dimensional plane. How do we find a circumscribed rectangle of $S$ with the minimum area? This is a basic problem in computational geometry with applications in computer graphics and robotics (see, e.g., Schneider and Eberly [66], Freeman and Shapiro [28] and Toussaint [72]). In this chapter, we investigate this problem from the viewpoint of computational complexity. More precisely, we assume that the boundary of set $S$ is a polynomial-time computable Jordan curve $\Gamma$ (i.e., $\Gamma$ has a polynomial-time representation $f : [0, 1] \to \mathbb{R}^2$), and ask what the complexity is of the minimum-area circumscribed rectangle of $S$.[1]

Let $\Gamma$ be a Jordan curve on the two-dimensional plane $\mathbb{R}^2$. For any line $L_\alpha$ that

[1] In the rest of the chapter, we write "the minimum circumscribed rectangle" to mean "the minimum-area circumscribed rectangle".

forms an angle $\alpha$ with the $x$-axis, there is a unique rectangle $R_\alpha$ that circumscribes the curve $\Gamma$ and has two of its four sides parallel to $L_\alpha$. Thus, the minimum circumscribed rectangle problem may be viewed as the problem of finding an angle $\alpha$ that minimizes the area of $R_\alpha$. This problem is similar to the general minimization problem studied in Ko [40], which asks for the minimum value of a polynomial-time computable real function. The main difference here is that the underlying function mapping $\alpha$ to rectangle $R_\alpha$ is not necessarily polynomial-time computable. Our main results, following this direction, can be summarized as follows:

(1) For a fixed polynomial-time computable angle $\alpha$, we can always find the circumscribed rectangle $R_\alpha$ of a polynomial-time computable Jordan curve $\Gamma$ in polynomial time if and only if $P = NP$.

(2) There exists a polynomial-time computable Jordan curve $\Gamma$ such that it has an uncountable number of minimum circumscribed rectangles, but none of them is computable (cf. the result on roots in Specker [67]).

(3) If a Jordan curve $\Gamma$ is polynomial-time computable, then the area $V$ of the minimum circumscribed rectangle $R_\alpha$ of $\Gamma$ is a right $\Sigma_2^P$-real number (See Section 2 for the definition).

(4) There exists a polynomial-time computable Jordan curve $\Gamma$ such that the problem of finding the area $V_{a,b}$ of the minimum circumscribed rectangle $R_\alpha$ of $\Gamma$, with the restriction of $a \leq \alpha \leq b$, is $\Sigma_2^P$-hard.

In addition to these results, we also study the problem of finding the minimum circumscribed square of a Jordan curve $\Gamma$. It is interesting to point out that this problem is not quite the same as the problem of finding the minimum circumscribed rectangle of $\Gamma$. In fact, a minimum square that encloses a Jordan curve $\Gamma$ is not necessarily a circumscribed square of $\Gamma$. In addition to the extension of result (2) above for the minimum circumscribed squares, we also show that for any polynomial-time computable Jordan curve $\Gamma$, there must exist at least one computable circumscribed

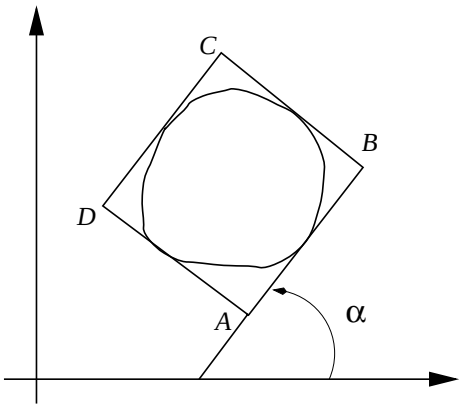


**Figure 6.1:** A circumscribed rectangle of a Jordan curve

square (not necessarily of the minimum area), but this square may have arbitrarily high complexity.

## 6.2 Minimum Circumscribed Rectangles

We first give a formal definition of circumscribed polygons of a Jordan curve.

**Definition 6.2.1** *We say a polygon $T$ circumscribes a Jordan curve $\Gamma$ if* (1) *$\Gamma$ intersects every side of $T$; and* (2) *every point of $\Gamma$ lies in the interior of $T$ or on $T$.*

As pointed out in Section 1, for a Jordan curve $\Gamma$ and a fixed angle $\alpha \in [0, \pi/2)$, there is a unique rectangle $R_\alpha$ that circumscribes the curve $\Gamma$ and has two of its four sides forming angle $\alpha$ with the $x$-axis (see Figure 6.1); we call $R_\alpha$ the circumscribed rectangle of $\Gamma$ at angle $\alpha$. We first consider the computational complexity of the rectangle $R_\alpha$, when $\alpha$ is a fixed polynomial-time computable real number. We say a rectangle $R$ is *polynomial-time computable* if its four corners are polynomial-time computable complex numbers.

Without loss of generality, we may assume that the circumscribed rectangle under consideration is rectangle $R_0$ at angle $\alpha = 0$, which has two horizontal and two

vertical sides. We note that the top horizontal side of this rectangle $R_0$ is $y = u$, where $u$ is the maximum $y$-value of the curve $\Gamma$, and the bottom horizontal line of $R_0$ is $y = b$, where $b$ is the minimum $y$-value of the curve $\Gamma$. Since the curve $\Gamma$ is polynomial-time computable, the values $u$ and $b$ are just the maximum and minimum values of a polynomial-time computable function, respectively.

From the above observation, the characterization of maximum and minimum values of Proposition 2.4.9 can be applied to the complexity of the rectangle $R_0$.

**Theorem 6.2.2** (*a*) *Let $\Gamma$ be a polynomial-time computable Jordan curve, and $R_0$ the circumscribed rectangle of $\Gamma$ at angle $\alpha = 0$. Let the four sides of the rectangle $R_0$ be $y = u$, $y = b$, $x = \ell$ and $x = r$, with $u > b$ and $r > \ell$. Then, $u$ and $r$ are left NP-real numbers, and $b$ and $\ell$ are right NP-real numbers.*

(*b*) *For any left NP-real number $u$, there is a polynomial-time computable Jordan curve $\Gamma$ such that the top horizontal side of its circumscribed rectangle $R_0$ at angle $\alpha = 0$ is $y = u$.*

*Proof.* (a) Let $f$ be a polynomial-time computable function that represents a Jordan curve $\Gamma$, and $f_x(t)$, $f_y(t)$ be two functions mapping $[0,1]$ to $\mathbb{R}$ such that $f(t) = \langle f_x(t), f_y(t)\rangle$ for $t \in [0,1]$. Then, both $f_x$ and $f_y$ are polynomial-time computable. It is clear that the top side of $R_0$ is $y = \max_{t\in[0,1]} f_y(t)$, and the bottom side of $R_0$ is $y = \min_{t\in[0,1]} f_y(t)$. Similarly, the right side of $R_0$ is $x = \max_{t\in[0,1]} f_x(t)$, and the left side of $R_0$ is $x = \min_{t\in[0,1]} f_x(t)$. Note that, for any function $g : [0,1] \to \mathbb{R}$, $\min_{t\in[0,1]} g(t) = -\max_{t\in[0,1]}(-g(t))$. Also note that, for any left cut $L$ of a real number $x$, the set $\{-d \mid d \in L\}$ is a right cut of $-x$. Thus, part (a) of the theorem follows from Proposition 2.4.9.

(b) Assume that $u > 0$. Let $g : [0,1] \to \mathbb{R}$ be a polynomial-time computable function with $\max_{t\in[0,1]} g(t) = u$, as given by Proposition 2.4.9. Without loss of generality, we may assume that (a) $g(0) = g(1) = 0$, and (b) $g(t) > 0$ for all $t \in (0,1)$.

Define a function $f : [0,1] \to \mathbb{R}^2$ as follows: $f$ on $[0,1/2]$ is the line segment from the point $\langle 1,0\rangle$ to the point $\langle 0,0\rangle$; and $f(t) = \langle 2t-1, g(2t-1)\rangle$ for $t \in (1/2,1]$. Then, it is clear that $f$ defines a polynomial-time computable Jordan curve $\Gamma$ and the circumscribed rectangle $R_0$ of $\Gamma$ at angle $\alpha = 0$ is formed by the following four lines: $y = u$, $y = 0$, $x = 1$, and $x = 0$. □

A set $A \subseteq \{0\}^*$ of strings formed by a singleton alphabet is called a *tally set*. Let $P_1$ and $NP_1$ denote the class of tally sets in $P$ and $NP$, respectively. It is known that if $P_1 \neq NP_1$ then there exists a left $NP$-real number which is not polynomial-time computable (see Ko [40]).

**Corollary 6.2.3** *In the following,* $(a) \Rightarrow (b) \Rightarrow (c)$*:*

$(a)$ $P = NP$.

$(b)$ *For every polynomial-time computable Jordan curve, its circumscribed rectangle* $R_0$ *at angle* $\alpha = 0$ *is polynomial-time computable.*

$(c)$ $P_1 = NP_1$.

**Corollary 6.2.4** $(a)$ *For any polynomial-time computable Jordan curve* $\Gamma$ *and any polynomial-time computable real number* $\alpha \in [0, \pi/2)$*, the height, width, and area of the circumscribed rectangle* $R_\alpha$ *of* $\Gamma$ *at angle* $\alpha$ *are left NP-real numbers.*

$(b)$ *If* $P_1 \neq NP_1$*, then there exists a polynomial-time computable Jordan curve* $\Gamma$ *such that the area of its circumscribed rectangle* $R_0$ *at angle* $\alpha = 0$ *is not polynomial-time computable.*

*Proof.* We note that the sum of two left $NP$-real numbers is still a left $NP$-real number, and the product of two positive left $NP$-real numbers is still a left $NP$-real number. □

We are interested in the minimum circumscribed rectangle of a Jordan curve $\Gamma$. Since the curve $\Gamma$ has a unique circumscribed rectangle $R_\alpha$ at each angle $\alpha \in [0, \pi/2)$,

this is essentially the problem of finding the angle $\alpha$ that minimizes the area of $R_\alpha$. Ko [40] pointed out that the computability of the maximum points of a computable function $g : [0, 1] \to \mathbb{R}$ is very similar to the computability of its roots. In particular, the result of Specker [67] about roots also holds for the maximum points; that is, there exists a computable function $g : [0, 1] \to \mathbb{R}$ such that it has an uncountable number of maximum points but none of them is computable. Furthermore, this result holds even if $g$ is required to be polynomial-time computable.

**Theorem 6.2.5** *There exists a polynomial-time computable Jordan curve* $\Gamma$ *such that* $\Gamma$ *has an uncountable number of minimum circumscribed rectangles but none of them is computable.*

*Sketch of Proof.* Since the proof follows the idea of Specker's theorem, we only present a sketch of the construction. We first let $\Gamma_0$ be the unit circle. That is, $\Gamma_0$ is the image of $f_0(\alpha) = \langle \cos\alpha, \sin\alpha \rangle$ on $[0, 2\pi]$. Note that all circumscribed rectangles of $\Gamma_0$ have the same area 4.

Now, let $\{x_n\}_{n=0}^{\infty}$ be a recursive enumeration of all computable real numbers in $[0, \pi/2]$. For each $n \geq 0$, we add to the circle $\Gamma_0$ a small $\Lambda$-shaped bump at angle $x_n$, of the same width and height $h_n = 2^{-k(n+t(n))}$, where $k \geq 2$ and $t(n)$ is the time required to enumerate the $n$-th number $x_n$ (see Figure 6.2). Let $\Gamma$ be the resulting curve. Since the height $h_n$ of the bump is smaller than $2^{-t(n)}$, the curve $\Gamma$ remains polynomial-time computable.

Let $\delta_n = \arccos(1/(1 + h_n))$. Then, the area of a circumscribed rectangle $R_\alpha$ of $\Gamma$ at angle $\alpha \in (x_n - \delta_n, x_n + \delta_n)$ is greater than 4. By choosing a sufficiently large $k$, we can ensure that the sum $\sum_{n\geq 0} 2\delta_n$ is less than $\pi/2$. Therefore, the set

$$T = [0, \pi/2] - \bigcup_{n\geq 0} (x_n - \delta_n, x_n + \delta_n)$$

is nonempty and has a positive measure. In addition, for each $\beta \in T$, the circumscribed rectangle $R_\beta$ of $\Gamma$ at angle $\beta$ remains the same with area 4. Finally, for

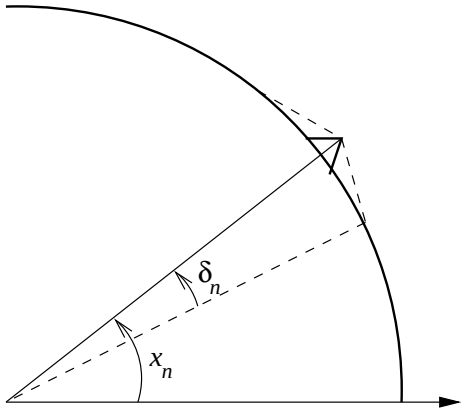


**Figure 6.2:** The construction of Theorem 6.2.5

each $\beta \in T$, one of the vertices of the circumscribed rectangle $R_\beta$ at angle $\beta$ is $\langle \sqrt{2} \cdot \cos(\beta - \pi/4), \sqrt{2} \cdot \sin(\beta - \pi/4) \rangle$, and so $R_\beta$ is not a computable rectangle. $\square$

Now we consider the problem of finding the minimum area of a circumscribed rectangle of a polynomial-time computable Jordan curve. First note that Theorem 6.2.2(a) can be extended so that the area $v(\alpha)$ of the circumscribed rectangle $R_\alpha$ of a polynomial-time computable Jordan curve at an angle $\alpha$ is actually an *NP-real function*. Thus, the problem of finding the minimum area of a circumscribed rectangle of $\Gamma$ is just to find the minimum value of an *NP*-real function. Proposition 2.4.9(b) suggests that the complexity of the minimum area $v(\alpha)$ is a right $\Sigma_2^P$-real number (i.e., a left $\Pi_2^P$-real number).

**Theorem 6.2.6** *Let $\Gamma$ be a polynomial-time computable Jordan curve. Then, for any dyadic rational numbers $0 \leq a < b \leq \pi/2$, the minimum area of a circumscribed rectangle $R_\alpha$ of $\Gamma$ at angle $\alpha \in [a, b]$ is a right $\Sigma_2^P$ real number.*

*Proof.* Assume that $\Gamma$ is computed by a real function $f : [0, 1] \to \mathbb{R}^2$ in time $p(n)$ for some polynomial $p$. Let $R_\alpha$ be a minimum-area circumscribed rectangle of $\Gamma$ at angle $\alpha$, and $v(\alpha)$ be its area. Without loss of generality, assume that $R_\alpha \subseteq [0, 1]^2$. Suppose $d$ is a dyadic rational in $\mathbb{D}_n$ which is greater than or equal to $v(\alpha)$. Let $e$ be a dyadic rational in $\mathbb{D}_{n+4}$ such that $|\alpha - e| \leq 2^{-(n+4)}$. Then, the area of the circumscribed

rectangle $R_e$ of $\Gamma$ at angle $e$ is less than $v(\alpha) + 4 \cdot 2^{-(n+4)}$. Let $S_e$ be a rectangle that is parallel to and encloses the rectangle $R_e$, with the following properties:

(a) The upper right corner of $S_e$ is a dyadic rational point $\mathbf{z}$ in $(\mathbb{D}_{n+5})^2$,

(b) The height $d_h$ and width $d_w$ of $S_e$ are two dyadic rationals in $\mathbb{D}_{n+5}$, and

(c) The distance between the corresponding sides of $R_e$ and $S_e$ is between $2^{-(n+5)}$ and $2^{-(n+4)}$.

Then, the area $S_e$ is $d_h \cdot d_w \leq area(R_e) + 4 \cdot 2^{-(n+4)} < d + 2^{-(n+1)}$.

We now design a $\Sigma_2^P$ machine (a polynomial-time nondeterministic oracle machine using a discrete oracle set $A \in NP$ as the oracle) $M$ to accept a right cut of $v(\alpha)$ based on the properties of rectangle $S_e$:

> Input: $d \in \mathbb{D}_n$.
>
> The machine $M$ first nondeterministically guesses a dyadic rational point $\mathbf{z}$ in $(\mathbb{D}_{n+5})^2$, a dyadic rational $e \in \mathbb{D}_{n+4}$ and two dyadic rationals $d_h, d_w$ in $\mathbb{D}_{n+5}$. Then, $M$ forms the rectangle $S_e$ from $\mathbf{z}, e, d_h, d_w$ as discussed above, and verifies that the curve $\Gamma$ is inside $S_e$. More precisely, the verification can be done as follows: For every $t \in [0,1] \cap \mathbb{D}_{p(n+5)}$, get a dyadic point $y_t$ that is within distance $2^{-(n+5)}$ of the point $f(t)$, and verify that $y_t$ lies inside the rectangle $S_e$ (note that $f(t) \in R_e \Rightarrow y_t \in S_e$ if property (c) above holds). In other words, $M$ uses the set $A = \{\langle \mathbf{z}, e, , d_h, d_w \rangle : (\exists t \in \mathbb{D}_{p(n+5)})\, y_t \notin S_e\}$ as an oracle, and accepts $d$ if $\langle \mathbf{z}, e, d_h, d_w \rangle \notin A$ and $d_h d_w < d + 2^{-(n+1)}$.

It is clear that the rectangle $S_e$ is uniquely defined by $\mathbf{z}, e, d_h$ and $d_w$, and it can be determined in polynomial time whether a dyadic point $\mathbf{x}$ is in $S_e$ or not. Thus, set $A$ is in $NP$, and $M$ is a $\Sigma_2^P$ machine.

The analysis given above shows that this $\Sigma_2^P$ machine accepts $d$ if $d \geq v(\alpha)$. In addition, if $M$ accepts $d$, then there must be a rectangle $S_e$ of area $d_h d_w < d + 2^{-(n+1)}$ such that all points in $\Gamma$ are either inside $S_e$ or within distance $2^{-(n+5)}$ of the boundary of $S_e$. So, the minimum area $v(\alpha)$ is less than $d + 2^{-(n+1)} + 4 \cdot 2^{-(n+5)} < d + 2^{-n}$. This means that if $d \leq v(\alpha) - 2^{-n}$, then $M$ rejects it. It follows that $M$ accepts a right cut of $v(\alpha)$. □

Next we consider the converse of the above theorem. Let $\Gamma$ be a polynomial-time computable Jordan curve. We will show that the general question of finding the minimum area of a circumscribed rectangle of $\Gamma$ at an angle between a given range $[a, b]$ is $\Sigma_2^P$-hard.

Recall that a set $A \subseteq \{0,1\}^*$ is in $\Sigma_2^P$ if and only if there exist a polynomial function $p$ and a polynomial-time computable predicate $Q$ such that, for all $w \in \{0,1\}^*$,

$$w \in A \Leftrightarrow (\exists u, \ell(u) = p(\ell(w)))(\forall v, \ell(v) = p(\ell(w)))Q(w, u, v). \tag{3.1}$$

For any $w \in \{0,1\}^+$, we define a dyadic rational number $x_w \in [0,1]$ as follows: Suppose $\ell(w) = n$, let $i_w$ be the integer whose $n$-bit binary representation is equal to $w$, and let $x_w = 1 - 2^{-(n-1)} + i_w \cdot 2^{-2n}$. In addition, we let $w'$ denote the successor of $w$ in the lexicographic order. Note that the interval $[x_w, x_{w'}]$ has length $2^{-2\ell(w)}$.

**Theorem 6.2.7** *Assume that $A \subseteq \{0,1\}^*$ is a set in $\Sigma_2^P$, and satisfies* (3.1)*. Let $h_n = \cos(2^{-(p(n)+2n+2)}\pi)$ for $n \in \mathbb{N}$, and for each $w \in \{0,1\}^+$, let $\alpha_w = x_w \pi/2$. Then, there exists a polynomial-time computable Jordan curve $\Gamma$ such that, for all $n \in \mathbb{N}$ and $w \in \{0,1\}^n$, the following holds:*

*(a) If $w \in A$ then $\min_{\alpha_w \leq \alpha \leq \alpha_{w'}} v(\alpha) = 2 + 2h_n$, and*

*(b) If $w \notin A$ then $\min_{\alpha_w \leq \alpha \leq \alpha_{w'}} v(\alpha) \geq 2 + 2h_n(1 + \delta_n)$,*

*where $v(t)$ is the area of the circumscribed rectangle $R_t$ of $\Gamma$ at angle $t$, and $\delta_n = 2^{-q(n)}$ for some polynomial function $q$.*

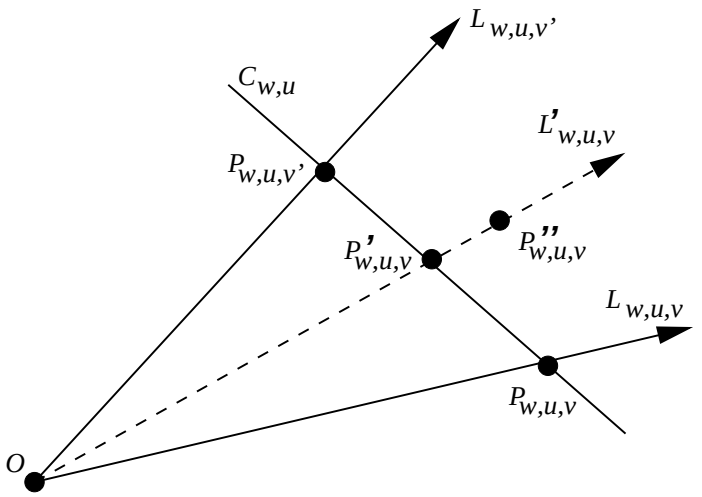


**Figure 6.3:** Points on the chord $C_{w,u}$.

*Proof.* The Jordan curve $\Gamma$ is to be constructed from a unit circle $\Gamma_0$ with center $O = \langle 0, 0\rangle$ and radius 1. The curve $\Gamma$ is identical to $\Gamma_0$ on the second, third and fourth quadrants. That is, we will define a function $f : [0, 1] \to \mathbb{R}^2$ to represent the curve $\Gamma$, and for $t \in [1/4, 1]$, $f(t) = \langle \cos(2t\pi), \sin(2t\pi)\rangle$.

Now we define $f$ on $[0, 1/4]$. For any $n \in \mathbb{N}$ and $w \in \{0, 1\}^n$, divide equally the interval $I_w = [x_w/4, x_{w'}/4]$ into $2^{p(n)}$ subintervals, with each one corresponding to a string $u \in \{0, 1\}^{p(n)}$ (following the lexicographic order), denoted $I_{w,u}$. So the length of the interval $I_{w,u}$ is $2^{-(p(n)+2n+2)}$. Similarly, divide $I_{w,u}$ into $2^{p(n)}$ subintervals of equal length, with each one corresponding to a string $v \in \{0, 1\}^{p(n)}$ and denoted $I_{w,u,v}$. So the length of an interval $I_{w,u,v}$ is $2^{-(2p(n)+2n+2)}$. Let $x_{w,u} = x_w/4 + i_u \cdot 2^{-(p(n)+2n+2)}$ and $x_{w,u,v} = x_{w,u} + i_v \cdot 2^{-(2p(n)+2n+2)}$, then $I_{w,u} = [x_{w,u}, x_{w,u'}]$ and $I_{w,u,v} = [x_{w,u,v}, x_{w,u,v'}]$. Let $P_{w,u}$ denote the point $\langle \cos(2x_{w,u}\pi), \sin(2x_{w,u}\pi)\rangle$. Let $C_{w,u}$ be the chord connecting the points $P_{w,u}$ and $P_{w,u'}$.

Also let $L_{w,u,v}$ be the half-line from the origin $\langle 0, 0\rangle$ of angle $2x_{w,u,v}\pi$, and $L'_{w,u,v}$ the half-line from the origin of angle $(x_{w,u,v} + x_{w,u,v'})\pi$. Let $P_{w,u,v}$ denote the intersection point of the half-line $L_{w,u,v}$ and the chord $C_{w,u}$, and $P'_{w,u,v}$ the intersection point of the half-line $L'_{w,u,v}$ and the chord $C_{w,u}$. Finally, let $P''_{w,u,v}$ be the point on the half-

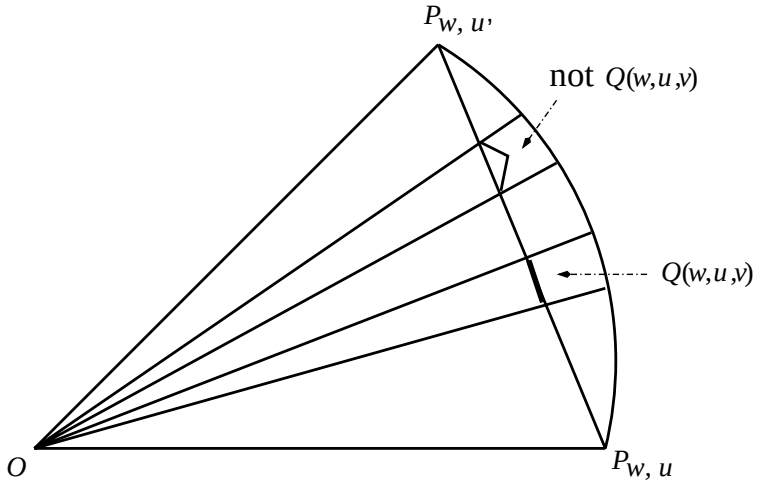


**Figure 6.4:** The function $f$ on $I_{w,u}$

line $L'_{w,u,v}$ with distance to the origin equal to the maximum of $length(\overline{OP_{w,u,v}})$ and $length(\overline{OP_{w,u,v'}})$ (see Figure 6.3).

We claim that the distance between $P'_{w.u.v}$ and $P''_{w,u,v}$ is at least $h_n \cdot 2^{-q(n)}$ for some polynomial function $q$. To see this, let us assume that $length(\overline{OP_{w,u,v}}) > length(\overline{OP_{w,u,v'}})$. Then, $length(\overline{OP''_{w,u,v}})$ equals $length(\overline{OP_{w,u,v}})$, which is at least $length(\overline{OP'_{w,u,v}})/\cos(2^{-(2p(n)+2n+2)}\pi)$. The claim follows now from the observation that $length(\overline{OP'_{w,u,v}}) > h_n$ and $1/\cos\alpha - 1 \geq \alpha^2/2$ when $\alpha \in [0, \pi/2)$.

Now, we define function $f$ on $I_{w,u,v} = [x_{w,u,v}, x_{w,u,v'}]$ as follows (see Figure 6.4):

(1) If $Q(w,u,v)$,then function $f$ is linear on $I_{w,u,v}$ with $f(x_{w,u,v}) = P_{w,u,v}$ and $f(x_{w,u,v'}) = P_{w,u,v'}$.

(2) If $\neg Q(w,u,v)$, then function $f$ is piecewise linear on $I_{w,u,v}$ with three breakpoints $f(x_{w,u,v}) = P_{w,u,v}$, $f(x_{w,u,v'}) = P_{w,u,v'}$, and $f((x_{w,u,v} + x_{w,u,v'})/2) = \langle m\cos\beta, m\sin\beta\rangle = P''_{w,u,v}$, where $m = \max\{length(\overline{OP_{w,u,v}}), length(\overline{OP_{w,u,v'}})\}$, and $\beta = (x_{w,u,v} + x_{w,u,v'})\pi$.

It is not hard to see that $f$ has a polynomial modulus function and is polynomial-time computable. We omit the details.

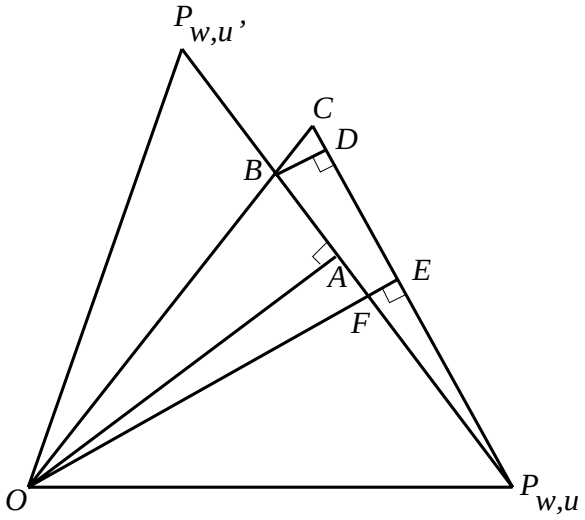


**Figure 6.5:** Proof of condition (b)

We now check conditions (a) and (b). First, we note that the bumps of $\Gamma$ all lie within the unit circle $\Gamma_0$. Therefore, the circumscribed rectangle of $\Gamma$ at an angle $\alpha$ in $[x_{w,u}, x_{w,u'}]$ must touch a point of $\Gamma$ in this angle.

(a) If $w \in \{0,1\}^n \cap A$, then $(\exists u, \ell(u) = p(n))(\forall v, \ell(v) = p(n))Q(w,u,v)$. According to the definition of function $f$, $f$ on $I_{w,u}$ is a line segment with a distance of $\cos(2^{-(p(n)+2n+2)}\pi) = h_n$ from the origin (see Figure 6.4). Therefore, the minimum area $v(\alpha)$ for $\alpha \in [\alpha_w, \alpha_{w'}]$ is $2(1+h_n) = 2+2h_n$.

(b) If $w \in \{0,1\}^n - A$, then $(\forall u, \ell(u) = p(n))(\exists v, \ell(v) = p(n))\neg Q(w,u,v)$. Thus, on each $I_{w,u}$, $f$ has at least a bump at $I_{w,u,v}$ (see Figure 6.4), with the tip of the bump of distance at least $h_n \cdot 2^{-q(n)}$ to $P'_{w,u,v}$. This implies that, for any line tangent to the portion of $\Gamma$ between $P_{w,u}$ and $P_{w,u'}$, its distance to the origin is at least $h_n(1+2^{-(q(n)+2)})$. This can be seen from Figure 6.5: Let $B$ denote $P'_{w,u,v}$ and $C$ denote $P''_{w,u,v}$. Suppose the line segment $\overline{BC}$ has length $\epsilon$, then $length(\overline{EF}) \geq 1/2 \cdot length(\overline{BD}) \geq 1/4 \cdot length(\overline{BC}) = \epsilon/4$. Therefore, the line $\overline{CP_{w,u}}$ has distance at least $length(\overline{OA}) + \epsilon/4 = h_n + \epsilon/4$ to the origin. □

**Corollary 6.2.8** *Assume that NP ≠ coNP. Then, there exists a polynomial-time computable Jordan curve* $\Gamma$*, such that the function* $v(a,b) = \min_{a \leq \alpha \leq b} v(\alpha)$ *is not com-*

*putable by an NP oracle Turing machine, where $v(\alpha)$ is the area of the circumscribed rectangle of $\Gamma$ at angle $\alpha$.*

In the above construction, we embedded each question of whether $w \in A$ for a given $A \in \Sigma_2^P$ in a different angle of the curve $\Gamma$. It remains open whether we can embed them at a single angle. That is, the question of whether every right $\Sigma_2^P$-real number is equal to the minimum area of a circumscribed rectangle $R_\alpha$ of a polynomial-time computable Jordan curve $\Gamma$ (without any restriction on the angle $\alpha$) is left open.

## 6.3 Circumscribed Squares

It is not hard to see that the problem of computing the minimum area of a square enclosing a polynomial-time computable Jordan curve $\Gamma$ is similar to that of a rectangle. Namely, we can guess the corners of a square and verify that they form a square and that every point of $\Gamma$ is within the square. Therefore, the minimum area of an enclosing square of $\Gamma$ is a right $\Sigma_2^P$-real number. In addition, with a construction that is only slightly different from that in the proof of Theorem 6.2.7, we can get results similar to Theorem 6.2.7 and Corollary 6.2.8.

It is important, however, to point out that this minimum square does not necessarily circumscribe the curve $\Gamma$. In fact, the mapping from a Jordan curve $\Gamma$ to the area of its minimum enclosing square is a continuous function (with respect to the hausdorff distance between Jordan curves). However, the mapping from a curve $\Gamma$ to its minimum circumscribed square is not a continuous function. This can be seen from the following simple example: The minimum circumscribed square of a square is itself, but the minimum circumscribed square of a rectangle of unequal sides is the square that forms a 45-degree angle with the rectangle (see Figure 6.6).

Thus, finding the minimum circumscribed square of a Jordan curve is a different question. Indeed, it is not immediately clear whether a Jordan curve must have a circumscribed square at all. It turns out that this question has an affirmative answer. Indeed, we can prove, by the intermediate value theorem that, for every Jordan curve $\Gamma$, there must exist at least one circumscribed square: For each $\alpha \in [0, \pi/2]$, let $R_\alpha$ denote the circumscribed rectangle of $\Gamma$ at angle $\alpha$, and let $a$ be the length of the side of the angle $\alpha$ with the $x$-axis, and $b$ the length of one of its neighboring side, and let $g(\alpha) = a - b$. Then, it is clear that $g$ is continuous on $[0, \pi/2]$ and $g(0) = -g(\pi/2)$. So, by the intermediate value theorem, there exists an $\alpha$ in $[0, \pi/2)$ such that $g(\alpha) = 0$, and $R_\alpha$ is a circumscribed square of $\Gamma$.

It is well known that the intermediate value theorem has an effective proof. Namely, for any computable function $f : [0, 1] \to \mathbb{R}$ with $f(0) < 0 < f(1)$, there exists at least one computable point $x \in (0, 1)$ such that $f(x) = 0$.

Since the above function $g$ is a computable function (actually, the difference of two *NP* real functions), it must have a computable root.

**Proposition 6.3.1** *Every polynomial-time computable Jordan curve* $\Gamma$ *on* $\mathbb{R}^2$ *has at least one computable circumscribed square.*

For the complexity of the minimum circumscribed square, we first note that the minimum circumscribed rectangles of the curve $\Gamma$ constructed in the proof of Theorem

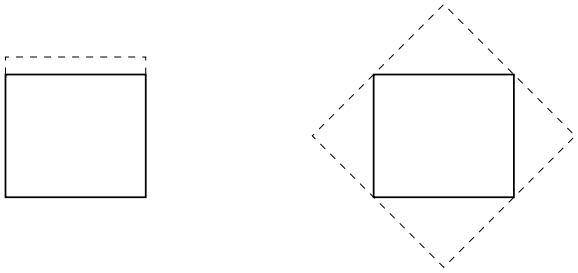

**Figure 6.6:** The minimum enclosing square and minimum circumscribed square of a rectangle of uneven sides

6.2.5 are actually all squares. Therefore, we see that a polynomial-time computable Jordan curve may not have a computable minimum circumscribed square.

On the other hand, we note that if the curve $\Gamma$ has a unique circumscribed square then, by Proposition 6.3.1, it must be computable. In this case, what is the complexity of the square? We can answer this question again by way of the intermediate value theorem. We recall the following theorem about the complexity of the intermediate value theorem.

**Proposition 6.3.2** ([*40, Theorem* 4.4]) *For any recursive real number $x \in [0,1]$, there exists a strictly increasing, polynomial-time computable function $h : [0,1] \to \mathbb{R}$ such that $x$ is the unique root of $h$ in $[0,1]$.*

A similar result holds for the unique circumscribed squares.

**Theorem 6.3.3** *For any recursive real number $\alpha \in [0, \pi/2]$, there exists a polynomial-time computable Jordan curve $\Gamma$ such that its circumscribed rectangle $R_\alpha$ at angle $\alpha$ is its unique circumscribed square.*

*Sketch of Proof.* Without loss of generality, we assume that $\alpha \in (\pi/8, \pi/4)$. Let $\Gamma_0$ be the circle with center $\langle 0, 0\rangle$ and radius 1. We construct the Jordan curve $\Gamma$ from $\Gamma_0$ by shrinking the portion of $\Gamma_0$ in the first quadrant to the right of angle $\alpha$ inward, and enlarging the portion of $\Gamma_0$ in the first quadrant to the left of angle $\alpha$ (see Figure 6.7).

More precisely, we first construct, as in the proof of Proposition 6.3.2, a polynomial-time computable, piecewise linear function $h : [0, \pi/2] \to \mathbb{R}$ with the following properties:

(i) $h$ is strictly increasing on $[0, \pi/2]$.

(ii) $|h(x)| \leq 1$ for all $x \in [0, \pi/2]$.

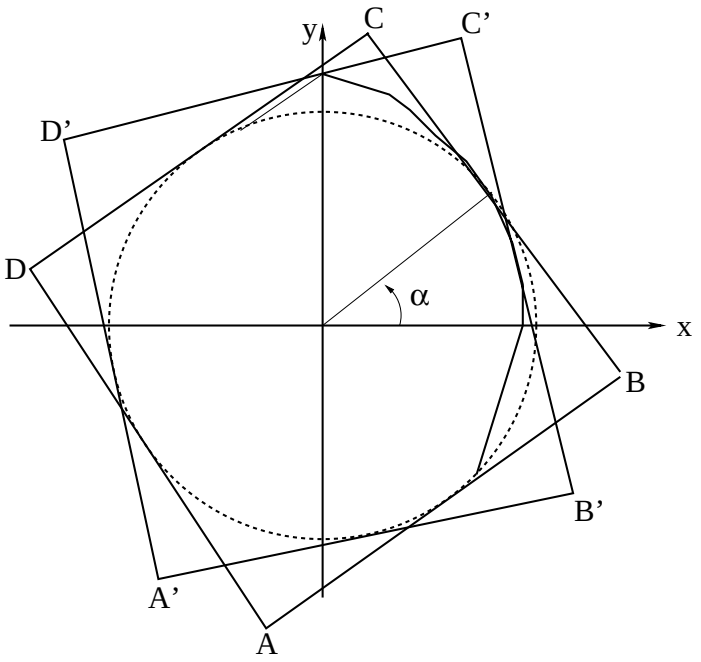


**Figure 6.7:** The function $f$

(iii) $\alpha$ is the unique root of $h$ on $[0, \pi/2]$.

(iv) $h(x) \leq 1/\cos(x - \alpha) - 1$ for all $x \in [0, \pi/2]$, and $h(\pi/2) \leq 1/\cos(\alpha) - 1$.

We then define a function $f : [0, 2\pi] \to \mathbb{R}^2$ (as the representation of $\Gamma$) as follows:

(i) For $x \in [0, \pi/2]$, $f(x) = \langle (1 + h(x)) \cos x, (1 + h(x)) \sin x \rangle$ (i.e., on the first quadrant, $\Gamma$ differs from $\Gamma_0$ by the amount of $h(x)$).

(ii) $f$ is linear on $[\pi/2, 5\pi/8]$ with $f(\pi/2) = \langle 0, 1+h(1) \rangle$ and $f(5\pi/8) = \langle \cos(5\pi/8), \sin(5\pi/8) \rangle$.

(iii) For $x \in [5\pi/8, 7\pi/4]$, $f(t) = \langle \cos x, \sin x \rangle$ (i.e., $\Gamma$ is identical to $\Gamma_0$ on interval $[5\pi/8, 7\pi/4]$).

(iv) $f$ is linear on $[7\pi/4, 2\pi]$ with $f(7\pi/4) = \langle \cos(7\pi/4), \sin(7\pi/4) \rangle$ and $f(2\pi) = \langle 1 + h(0), 0 \rangle$.

This design makes all circumscribed rectangles $R_\beta$ of $\Gamma$ at an angle $\beta < \alpha$ have negative $g(\beta)$ values, and those $R_\beta$ with $\beta > \alpha$ have positive $g(\beta)$ values, where $g$

is the difference between two neighboring sides of $R_\beta$ as defined earlier. Note that property (iv) of function $h$ ensures that $R_\alpha$ is a square. Thus, $R_\alpha$ is the unique circumscribed square of $\Gamma$. □

We say a function $t(n)$ is a *fully-time constructible function* if there exists a Turing machine $M$ that halts on input $n$ in exactly $t(n)$ moves. Most familiar time bounds, such as $n^2$, $2^n$, are fully-time constructible (see, e.g., Du and Ko [26]).

**Corollary 6.3.4** *For any fully time-constructible function $t(n)$, there exists a $P$-computable Jordan curve $\Gamma$ which has a unique circumscribed square $S$ but $S$ is not computable in time $t(n)$.*

## 6.4 Remarks

In this chapter, we studied the computational complexity of finding, from a given polynomial-time computable Jordan curve, the circumscribed rectangles and squares of the minimum area. We applied the proof techniques for the general minimization problem to the minimum circumscribed rectangle problem, and showed results similar to the general minimization problem. In particular, we characterized the complexity of the area of the minimum circumscribed rectangle by the discrete complexity class $\Sigma_2^P$.

We note that, however, the known results about the general minimization problem cannot apply to our problem directly. Since we are dealing with geometric objects, the constructions have more constraints. In fact, for the minimum area of a circumscribed rectangle, there is a small gap between our upper bound (Theorem 6.2.6) and lower bound (Theorem 6.2.7). This is, as pointed out at the end of Section 3, because the extra constraints seem to interfere each other, we are not able to put the constructions around a single angle.

Furthermore, we remark that the results in Section 3 can be adapted to two other important concepts in computational geometry, namely, the minimum-perimeter circumscribed rectangles of a Jordan curve (see DePano[24] and Pirzadeh [62]) and the least width of a Jordan curve (see Houle and Toussaint [39] and Pirzadeh [62]).[2] We note that the perimeter and the width of a circumscribed rectangle $R_\alpha$ at a fixed angle $\alpha$ are, like the area of $R_\alpha$, left $NP$-real numbers. So, all the results from Corollary 6.2.4 to Corollary 6.2.8 also hold for these two concepts. It is interesting to point out that the maximum width of $\Gamma$, which is equal to the diameter of $\Gamma$,[3] is a left $NP$-real number, and hence has lower complexity than the least width of $\Gamma$, assuming that $NP \neq coNP$.

Finally, we discuss the differences between our results with those in computational geometry. Toussaint [1983] has shown that it takes $O(n)$ time to compute a minimum-area circumscribed rectangle of an $n$-sided polygon, while we have proved that the problem of finding minimum-area circumscribed rectangles of a polynomial-time computable Jordan curve is undecidable, and the problem of finding the minimum area of circumscribed rectangles of a polynomial-time computable Jordan curve is in $\Sigma_2^P$. This difference stems from the different computational models and complexity measures used in the two approaches. In the computational geometry approach, the curves to be studied are restricted to be polygons, and the input $n$-sided polygons are presented to the algorithm with the $n$ vertices given explicitly. In addition, the time complexity of the algorithm is measured against the size $n$ of the input polygon. In our approach, the algorithm needs to work on all polynomial-time computable curves, not just polygons, and the time complexity is measured with respect to the output

[2]The width of a Jordan curve $\Gamma$ at a given angle $\alpha$ is the distance of two parallel lines $L_1$ and $L_2$ at angle $\alpha$ such that every point of $\Gamma$ is between $L_1$ and $L_2$ or on $L_1 \cup L_2$; the least and greatest widths are the minimum and maximum of widths over all angles, respectively.

[3]The diameter of $\Gamma$ is the maximum length of line segments $\overline{AB}$ over all pairs $A, B \in \Gamma$.

precision of the circumscribed rectangle. We note that although the curve $\Gamma$ in our model may be approximated by polygons, an approximate polygon with error $\leq 2^{-n}$ would have $2^{p(n)}$ vertices for some polynomial $p$. When it is applied to this approximate polygon, Toussaint's algorithm would take exponential time (with respect to the precision $n$). In other words, our approach considers a wider range of problems, and the results are consistent with the results from computational geometry.

# Chapter 7

# The Pancake Problem

## 7.1 Introduction

Given two Lebesgue measurable sets of arbitrary shapes in the two-dimensional plane $\mathbb{R}^2$, there exists a line that simultaneously bisects them (i.e., the line cuts each set into two parts of equal area). This is the famous *Pancake Theorem*, or, the two-dimensional version of a more general *Ham Sandwich Theorem*. These theorems are related to two fundamental theorems, the *Brouwer Fixed Point Theorem* and the *Borsuk-Ulam Theorem*, in topology (see, e.g., Fulton [30]).

The Pancake Theorem leads to a computational problem, called the *pancake problem*, of finding the line, called the *bisector* (or *pancake/ham sandwich cut* in the literature), that simultaneously bisects two given regions. In this chapter, we study the computational complexity of the pancake problem. It has been well studied for the cases where the sets in question are polygons (see, e.g., [1; 68]), or are composed of a finite number of points (in this case, the measure is the number of points instead of the area; see, e.g., [27; 50; 51]). Here, we consider the general cases where the sets may be subsets of the plane $\mathbb{R}^2$ of more complicated shapes and apply the model of Turing-machine based complexity theory of real functions to this problem.

We also consider two related problems: (1) for a fixed angle $\alpha \in [0, \pi)$ (e.g., $\alpha = \pi/2$), computing the bisector $L$ of a set $S$ such that the angle from the positive $x$-axis to $L$ is $\alpha$; and (2) computing two lines that are perpendicular to each other and divide a set $S$ into four parts of equal area. Our main results use discrete complexity class $\#P$ and other related classes to characterize the complexity of the bisectors. They can be summarized as follows:

(1) These problems are all solvable if the sets in question are polynomial-time approximable and there exists a unique bisector (or a unique pair of lines for the problem of dividing a set into four parts). However, there are no fixed complexity bounds even if the sets have sufficient polynomial-time representations. For example, it can be arbitrarily hard to compute two lines that are perpendicular to each other and divide a polynomial-time approximable set $S$ into four parts of equal area, even if $S$ is also polynomial-time recognizable and convex, and has a polynomial-time computable Jordan curve as the boundary.

(2) The complexity of finding the vertical bisector $L$ of a polynomial-time approximable set $S$ has a lower bound of $P^{\#P_1[1]}$ and an upper bound of $P^{\#P}$, provided that $S$ is *sufficiently thick* around $L$. Here $P^{\#P}$ is the class of languages $A$ such that $A$ is polynomial-time decidable with a $\#P$ function as an oracle, $\#P_1$ is the unary version of $\#P$, and the notation [1] in $P^{\#P_1[1]}$ means that the oracle is only queried once.

(3) The complexity of finding the bisector $L$ that bisects two $P$-approximable sets $S_1$ and $S_2$ has a lower bound of $P^{\#P_1[1]}$ and an upper bound of $P^{\#P}$, provided that $S_1$ and $S_2$ are linearly separable (i.e., $S_1$ and $S_2$ are on different sides of some line) and both $S_1$ and $S_2$ are *sufficiently thick* around their own bisectors at all angles.

## 7.2 Bisecting One Set

From now on, unless specified otherwise, by "bisecting" a set we mean "dividing" a subset of $\mathbb{R}^2$ into two parts of equal area using a line, and by "bisector" of a set we mean a line that bisects the set. The Pancake Theorem states that two bounded sets have a common bisector. We first study the complexity of bisecting one set.

Let $S$ be a bounded set in $\mathbb{R}^2$. Fix an angle $\alpha \in [0, \pi)$. If the angle from the positive $x$-axis to a line $L$ is $\alpha$, we say that $L$ is at angle $\alpha$. It is easy to see that there exists a bisector $L$ of $S$ at each angle $\alpha$. It is possible that there is another bisector $L'$ of $S$ at the same angle $\alpha$. In this case, the area of the part of $S$ between $L$ and $L'$ is zero, and any line between $L$ and $L'$ that is parallel to $L$ is a bisector of $S$ at angle $\alpha$. Then, the complexity of some bisector can be very high. We assume that this does not happen and assume that there exists a unique bisector of $S$ at any fixed angle $\alpha$.

**Definition 7.2.1** *Let $S$ be a bounded set in $\mathbb{R}^2$. A line $L$ defined by $x = a$ divides $S$ into two parts, with the left part denoted $S_{x \le a}$ and the right part $S_{x \ge a}$. We define the* thickness *of $S$ at $L$, denoted $thk_S(L)$, as the greatest lower limit of the area change of $S_{x \le a}$ when moving $L$ parallelly, that is,*

$$thk_S(L) = \liminf_{\delta \to 0} \frac{area(S_{x \le a+\delta}) - area(S_{x \le a})}{\delta}.$$

*For a line $L$ at an angle $\alpha \in [0, \pi)$, we can also define $thk_S(L)$ in a similar way. More precisely, we rotate $S$ and $L$ about the origin by an angle $\pi/2 - \alpha$ to obtain $S'$ and $L'$, respectively, and define $thk_S(L) = thk_{S'}(L')$.*

It is obvious that for any line $L$ at any angle $\alpha \in [0, \pi)$, $thk_S(L)$ is nonnegative. If $S$ is a convex Jordan domain, then $thk_S(L)$ is the length of the chord $L \cap S$. If $L$ is a bisector of $S$ at angle $\alpha$ such that $thk_S(L) > 0$, then $L$ is the unique bisector of $S$

at angle $\alpha$. Note that it is possible that $thk_S(L) > 0$, but for any $\epsilon > 0$, there exists a line $L'$ such that $L'$ is parallel to $L$ and $\text{dist}(\text{L}, \text{L}') < \epsilon$, but $thk_S(L') = 0$. We say $S$ has a positive thickness around a line $L$ if there exist two positive real numbers $\epsilon$ and $\delta$, such that if $L'$ is a line parallel to $L$ with $\text{dist}(\text{L}, \text{L}') < \epsilon$, then $thk_S(L') > \delta$.

If for a set $S$ and an angle $\alpha$, there are two bisectors $L$ and $L'$, then $thk_S(L) = thk_S(L') = 0$. However, it is possible that there exists a unique bisector $L$ of $S$ at angle $\alpha$ even if $thk_S(L) = 0$. Below we show that the complexity of finding such a bisector can be arbitrarily hard, while a positive thickness around a bisector reduces the complexity (see Theorem 7.2.3).

**Theorem 7.2.2** *Let $b \in (0, 1)$ be a computable number. There exists a $P$-appro -ximable/recognizable set $S \subseteq [0, 1]^2$ such that the line $L$ defined by $x = b$ is the unique bisector of $S$ at angle $\pi/2$; furthermore, $thk_S(L) = 0$ and for any line $L'$ defined by $x = b'$, where $b' \in (0, 1) - \{b\}$, $thk_S(L') > 0$.*

*Proof.* We construct a $P$-approximable/recognizable set $S$ such that (1) the boundary of $S$ is the union of line segment $\{\langle x, 0\rangle : x \in [0, 1]\}$ and the image of a polynomial-time computable function $f$, and is of a finite length; and (2) the set $S$ is "symmetric" with respect to the bisector.

We assume that $b$ is not a dyadic since, if $b$ is a dyadic, it is very easy to construct a polygon whose vertical bisector is defined by $x = b$. Let $\phi$ be a computable Cauchy function in $CF_b$ and let $t : \mathbb{N} \to \mathbb{N}$ be a computable function which bounds the runtime of the function $\phi$. We assume that $t(k) \geq 2k$ for all $k \in \mathbb{N}$. We inductively define two sequences $\{\underline{d}_k\}$ and $\{\overline{d}_k\}$ of dyadic numbers in $[0, 1]$:

$\underline{d}_1 = 0; \overline{d}_1 = 1;$

$\underline{d}_k = \max\{\underline{d}_{k-1}, \phi(k) - 2^{-k}\},$

$\overline{d}_k = \min\{\overline{d}_{k-1}, \phi(k) + 2^{-k}\}, \quad k \geq 2.$

We observe that for any $k \geq 2$,

$$0 = \underline{d}_1 \leq \underline{d}_2 \leq \underline{d}_k \leq \underline{d}_{k+1} < b < \overline{d}_{k+1} \leq \overline{d}_k \leq \overline{d}_2 \leq \overline{d}_1 = 1,$$

and that $\underline{d}_k$ and $\overline{d}_k$ are computable in time $O(s(k))$, where $s(k) = \sum_{i=1}^{k} t(i) \geq k^2 + k$.

Let $I_1 \subseteq \mathbb{N}$ be a set of indices such that $i \in I_1$ if and only if $\underline{d}_{i+1} > \underline{d}_i$; in other words, $\{\underline{d}_i\}_{i \in I_1}$ is the largest strictly increasing subsequence of the increasing subsequence $\{\underline{d}_i\}_{i \in \mathbb{N}}$. Similarly, let $I_2 \subseteq \mathbb{N}$ be a set of indices such that $i \in I_2$ if and only if $\overline{d}_{i+1} < \overline{d}_i$. Note that since $b$ is not a dyadic, both $I_1$ and $I_2$ contain infinitely many elements. Let $a_{m,n}$ be the $n$-th smallest number in $I_m$, where $m \in \{1, 2\}$ and $n \in \mathbb{N}$. For any $n \in \mathbb{N}$, let $i$ denote $a_{1,n}$ and $j$ denote $a_{1,n+1}$, $h_1 = \max(a_{1,n+1}, a_{2,n+1})$ and $h_2 = \max(a_{1,n+2}, a_{2,n+2})$. We note that $\underline{d}_j - \underline{d}_i \geq 2^{-j}$ since $\underline{d}_j - \underline{d}_i$ is a positive dyadic in $\mathbb{D}_j$. We call $[\underline{d}_i, \underline{d}_j]$ the $n$-th interval of the first kind. The part of set $S$ between two lines $x = \underline{d}_i$ and $x = \underline{d}_j$ is a pentagon of area $2^{-(2s(h_1+1)+1)}$; more precisely, the five vertices of the pentagon are $\langle \underline{d}_i, 0 \rangle$, $\langle \underline{d}_i, 2^{-2s(h_1+1)} \rangle$, $\langle (\underline{d}_i + \underline{d}_j)/2, 2^{-2s(h_1+1)}/(\underline{d}_j - \underline{d}_i) - (2^{-2s(h_1+1)} + 2^{-2s(h_2+1)})/2 \rangle$, $\langle \underline{d}_j, 2^{-2s(h_2+1)} \rangle$, and $\langle \underline{d}_j, 0 \rangle$; we denote this pentagon $P_{i,j,h_1,h_2}$. Similarly we define the other parts of $S$ according to $I_2$ (e.g., we will have the $n$-th interval of the second kind).

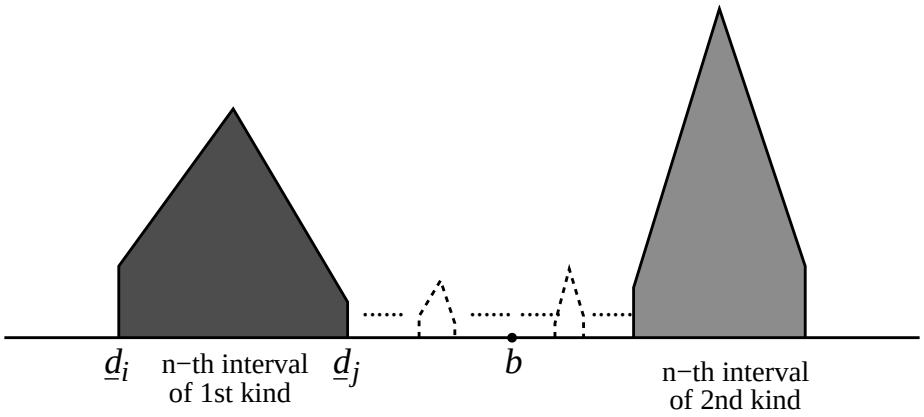


**Figure 7.1:** The set $S$ consisting of pentagons.

We can see that the line $x = b$ is the unique bisector of $S$ at angle $\pi/2$, since from the construction of $S$, for any $n \in \mathbb{N}$, the pentagon associated with the $n$-th

interval of the first kind and that associated with the $n$-th interval of the second kind are of the same area. Note that the boundary of $S$ is the union of the unit interval $[0,1]$ on the $x$-axis and the set $\{\langle x, f(x)\rangle : x \in [0,1]\}$, where the function $f$ is easy to obtain from the construction of $S$ above. We first show that $f$ is polynomial-time computable by proving that $f$ satisfies both conditions in Proposition 2.4.5. Indeed, it is easy to see that $f$ has a linear modulus of continuity since, for each $n \in \mathbb{N}$, let $i = a_{1,n}$, $j = a_{1,n+1}$, $h_1 = \max(a_{1,n+1}, a_{2,n+1})$ and $h_2 = \max(a_{1,n+2}, a_{2,n+2})$, then we have $s(h_1+1) \geq h_1^2 + 1 \geq j$, $|\underline{d}_j - \underline{d}_i| \geq 2^{-j} \geq 2^{-s(h_1+1)}$ and $\frac{2^{-2s(h_1+1)}/(\underline{d}_j - \underline{d}_i)}{(\underline{d}_j - \underline{d}_i)/2} < 2$, which implies that the derivative (which exists almost everywhere) of $f$ is between $-2$ and 2; furthermore, we can see that the boundary of $S$ is of a finite length. Next we show that for any two integers $m, n \in \mathbb{N}$ and any dyadic $d \in \mathbb{D}_m \cap [0,1]$, a dyadic $e$ can be computed in time $O(m+n)$ such that $|e - f(d)| < 2^{-n}$ by the following algorithm:

(1) Compute in time $O(n)$ an integer $k$ such that $k = \max\{i : s(i) \leq n\}$ by simulating a machine $M$ with time bound $t$ that computes $\phi$ as follows: Let $M$ compute one by one $\phi(1)$, $\phi(2)$, $\cdots$, and halt after $n$ moves, now $k$ is the maximum number such that $\phi(k)$ is computed in this process. (*Note that the simulation may have started to compute $\phi(k+1)$ but it is not finished since it is terminated after $n$ moves. In this case $s(k) \leq n < s(k+1)$.* )

(2) Compute in time $O(n)$ the (multi)set $\{\underline{d}_i, \overline{d}_i : i \leq k\}$, $I_1' := I_1 \cap \{1, 2, \cdots, k-1\}$, and $I_2' := I_2 \cap \{1, 2, \cdots, k-1\}$. (*Note that it may take more than $n$ moves to decide whether $k \in I_1$ (or $k \in I_2$), but we do not need this result. This step can be combined with step (1).*)

(3) Compute in time $O(k^2)$ three integers $q = \min(||I_1'||, ||I_2'||)$, $i_0 = a_{1,q}$ and $j_0 = a_{2,q}$. Check in time $O(m+k)$ whether $d \in [\underline{d}_{i_0}, \overline{d}_{j_0}]$, $d < \underline{d}_{i_0}$, or $d > \overline{d}_{j_0}$: if

$d \in [\underline{d}_{i_0}, \overline{d}_{j_0}]$, let $e = 0$ and halt; if $d < \underline{d}_{i_0}$, go to step (4); if $d > \overline{d}_{j_0}$, go to step (5).

(4) Compute in time $O(k^2 + m) = O(n + m)$ three integers $\ell < q$, $i = a_{1,\ell}$ and $j = a_{1,\ell+1}$ such that $d \in [\underline{d}_i, \underline{d}_j]$. There are two sub cases:

(4.1) $\ell = q - 1$. Compute in time $O(k^2)$ an integer $h_1 = \max(a_{1,q}, a_{2,q})$. We have $h_2 := \max(a_{1,q+1}, a_{2,q+1}) > k$ (we do not compute $h_2$ now) and $s(h_2) > n$. The quadrangle $P'_{i,j,h_1}$ with vertices $\langle \underline{d}_i, 0\rangle$, $\langle \underline{d}_i, 2^{-2s(h_1+1)}\rangle$, $\langle (\underline{d}_i + \underline{d}_j)/2, 2^{-2s(h_1+1)}/(\underline{d}_j - \underline{d}_i) - 2^{-2s(h_1+1)}/2\rangle$ and $\langle \underline{d}_j, 0\rangle$ is an approximation to the pentagon $P_{i,j,h_1,h_2}$ with error $\leq 2^{-2s(h_2+1)} < 2^{-2n}$. Do an interpolation in time $O(m + n + j + s(h_1 + 1)) = O(m + n)$ on $P'_{i,j,h_1}$ to obtain an approximation $e$ to $f(d)$ such that $|e - f(d)| < 2^{-n}$ and halt.

(4.2) $\ell < q - 1$. Compute in time $O(k^2)$ two integers $h_1 = \max(a_{1,\ell+1}, a_{2,\ell+1})$ and $h_2 := \max(a_{1,\ell+2}, a_{2,\ell+2})$. (*Now the pentagon $P_{i,j,h_1,h_2}$ is exactly the part of $S$ between lines $x = \underline{d}_i$ and $x = \underline{d}_j$.*) Do an interpolation in time $O(m + n + j + s(h_1 + 1) + s(h_2 + 1)) = O(m + n)$ on $P_{i,j,h_1,h_2}$ to obtain an approximation $e$ to $f(d)$ such that $|e - f(d)| < 2^{-n}$ and halt.

(5) (*It is similar to step (4) and is omitted.*)

We can see that the above algorithm takes time $O(m + n)$ since each step takes time $O(m + n)$. Also from the algorithm, for the cases of $d < \underline{d}_{i_0}$ and $d > \overline{d}_{j_0}$, an approximation $e$ to $f(d)$ with an error $\leq 2^{-n}$ is obtained. We need to show that if $d \in [\underline{d}_{i_0}, \overline{d}_{j_0}]$, $e = 0$ is an approximation to $f(d)$ with an error $< 2^{-n}$, that is, $|f(d)| < 2^{-n}$. Without loss of generality, assume that $d \in [\underline{d}_{i_0}, b)$, then there exists an integer $\ell \geq q$ such that $d \in [\underline{d}_i, \underline{d}_j]$, where $i = a_{1,\ell}$, $j = a_{1,\ell+1}$; that is, $d$ is in the $\ell$-th interval of the first kind. Let $h_1 = \max(a_{1,\ell+1}, a_{2,\ell+1})$ and $h_2 = \max(a_{1,\ell+2}, a_{2,\ell+2})$. From the definition of $q$, $h_2 > h_1 \geq k$. Then $f(d)$ is decided by the pentagon

$P_{i,j,h_1,h_2}$. Since $h_1 \geq k$, $s(h_2+1) > s(h_1+1) \geq s(k+1) > n$, and $|f((\underline{d}_i, \underline{d}_j)/2)| = 2^{-2s(h_1+1)}/(\underline{d}_j - \underline{d}_i) - (2^{-2s(h_1+1)} + 2^{-2s(h_2+1)})/2 < 2^{-n}$, we have $|f(d)| < 2^{-n}$, which completes the proof that $f$ is polynomial-time computable.

In order to check whether a point $\langle x, y \rangle$ is in $S$, we check whether $0 < y < f(x)$. As $f$ is polynomial-time computable, $S$ is $P$-recognizable. Note that $f(b) = 0$, and the boundary of $S$ is the union of two Jordan curves and of a finite length, it follows that $S$ is also $P$-approximable (Chou and Ko [18]). It is also easy to check that $thk_S(L) = 0$ and for any line $L'$ defined by $x = b'$, where $b' \in (0,1) - \{b\}$, $thk_S(L') > 0$: $f$ is continuous, thus for any line $L_r$ defined by $x = r$, where $r \in (0,1)$, $thk_S(L_r) = f(r)$; furthermore, $f(r) > 0$ iff $r \neq b$. □

Theorem 7.2.2 is a negative result for the case where the thickness of the set $S$ at a bisector is zero. Next we prove a positive result for the case where $S$ has a positive thickness around a bisector.

Let $S$ be a $P$-approximable set with a positive thickness around a bisector $L_\alpha$ at an angle $\alpha$. Without loss of generality, we consider the case $\alpha = \pi/2$.

**Theorem 7.2.3** *In the following,* $(a) \Rightarrow (b) \Rightarrow (c)$*:*

$(a)$ $FP = \#P$.

$(b)$ *For any $P$-approximable set $S \subseteq [0,1]^2$ that has a positive thickness $W > 0$ around the unique vertical bisector $L$ defined by $x = b$, the real number $b$ is polynomial-time computable.*

$(c)$ $FP_1 = \#P_1$.

*Proof.* (a) ⇒(b). We define a function $g : [0,1] \to \mathbb{R}^2$ such that $g(t) = area(S_{x \leq t}) - area(S_{x>t})$, where $S_{x \leq t} = \{\langle x, y \rangle \in S : x \leq t\}$ and $S_{x>t} = S - S_{x \leq t}$. Then $b$ is the root of $g(x)$. We will show that the function $g$ is polynomial-time computable under the condition $FP = \#P$. The function $g$ is increasing and furthermore, since $S$ has a positive thickness around the vertical bisector $L$, we have $|area(S_{x \leq b+\delta}) -$

$area(S_{x\leq b})| \geq |\delta| \cdot thk_S(L)/2$ when $\delta$ is sufficiently small, which means that $g$ has a polynomial inverse modulus near $b$. According to Corollary 4.7 of Ko [40], the root $b$ of $g(x)$ can be computed in polynomial time by binary search.

Now we show how to compute $g$ in polynomial time. First, $g$ has a linear modulus of continuity, because $S \subseteq [0,1]^2$ and by the definition of $g$, $|g(t_1)-g(t_2)| \leq |t_1-t_2|$ for $t_1, t_2 \in [0,1]$. Then by Proposition 2.4.5, we only need to show that the values of $g$ at dyadic points can be approximated in polynomial time. Since $S$ is $P$-approximable, there exists a polynomial-time oracle Turing machine $M$ such that for any input $n$, the error set $E_n(M)$ has size $\mu^*(E_n(M)) \leq 2^{-n}$. Let $p$ be a polynomial function that bounds $M$. Without loss of generality, assume that $p(n) > n$.

For $n \in \mathbb{N}$ and $t \in \mathbb{D}_n \cap [0,1]$, consider the following two sets:

$$A(n,t) = \{\langle d_1, d_2\rangle \in \mathbb{D}^2_{p(n)} : d_1 < t, M^{d_1,d_2}(n) = 1\};$$
$$B(n,t) = (\mathbb{D}_{p(n)} \cap [0,t)) \times (D_{p(n)} \cap [0,1]) - A(n,t).$$

For $x, y, d \in \mathbb{R}$ with $d > 0$, let $N(\langle x, y\rangle; d)$ denote the set $\{\langle x', y'\rangle \in \mathbb{R}^2 : |x'-x| < d, |y'-y| < d\}$. Suppose that a dyadic point $\mathbf{d} = \langle d_1, d_2\rangle$ is in $A(n,t)$, then for any $\mathbf{z} \in N(\mathbf{d}; 2^{-(p(n)+1)})$, since $\mathbf{z}$ has an oracle representation $\langle \phi, \psi\rangle$ such that $\phi(i) = b_{d_1}(i)$ and $\psi(i) = b_{d_2}(i)$ for all $i \leq p(n)$, the computation $M^{\phi,\psi}(n)$ works exactly the same as that of $M^{d_1,d_2}(n)$, and hence it outputs 1. That means that either $\mathbf{z} \in S_{x\leq t}$ or $\mathbf{z} \in E_n(M)$; in other words, $N(\mathbf{d}; 2^{-(p(n)+1)}) \subseteq S_{x\leq t} \cup E_n(M)$. Similarly, if $\mathbf{d} \in B(n,t)$, then $N(\mathbf{d}; 2^{-(p(n)+1)}) \subseteq S_{x\geq t} \cup E_n(M)$. Note that the small squares $N(\mathbf{d}; 2^{-(p(n)+1)})$, where $\mathbf{d} \in (\mathbb{D}_{p(n)} \cap [0,t)) \times (D_{p(n)} \cap [0,1])$, do not overlap each other and the union of them cover the rectangle $[-2^{-(p(n)+1)}, t - 2^{-(p(n)+1)}] \times [-2^{-(p(n)+1)}, 1 + 2^{-(p(n)+1)}]$ (except a finite number of line segments whose area is zero). Now it is easy to see that $2^{-2p(n)} \cdot ||A(n,t)||$ is close to the area of $S_{x\leq t}$, with an error bounded by $\mu^*(E_n(M)) + 4 \cdot 2^{-(p(n)+1)} < 2^{-(n-1)}$. We define a function $G : \mathbb{N} \times (\mathbb{D} \cap [0,1]) \to \mathbb{N}$ such that $G(n,t) = ||A(n,t')||$ for all $(n,t) \in \mathbb{N} \times (\mathbb{D} \cap [0,1])$, where $t' = \max\{x \in \mathbb{D}_n : x \leq t\}$. Note that $A(n,t')$ is the number of elements $\langle d_1, d_2\rangle \in \mathbb{D}^2_{p(n)}$ that satisfy

the polynomial-time computable predicate $d_1 < t' \wedge M^{d_1,d_2}(n) = 1$, thus according to Theorem 2.3.2 (c), $G$ is in $\#P$. Therefore, if $\#P = FP$, then the area of $S_{x \leq t}$ can be computed in polynomial time. The same conclusion holds for $S_{x>t}$. Therefore, $g$ is polynomial time computable if $\#P = FP$.

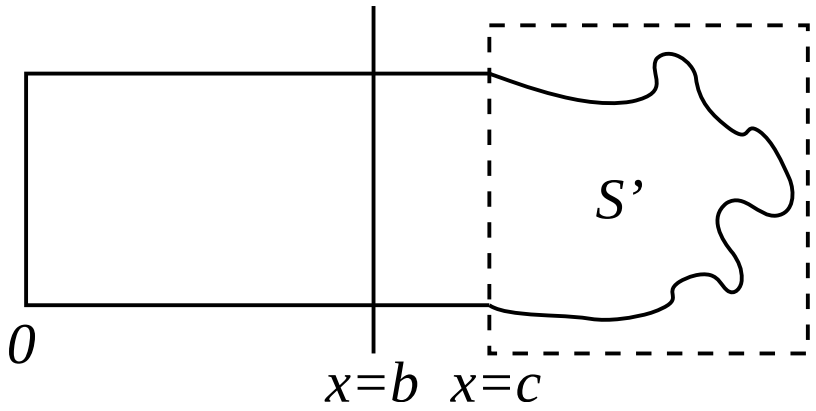


**Figure 7.2:** The domain $S$ for (b)⇒ (c) of Theorem 7.2.3.

(b)⇒ (c). Chou and Ko [18] constructed a $P$-approximable set $S' \subseteq [0,1]^2$ such that the area of $S'$ is polynomial time computable if and only if $\#P_1 = FP_1$. We construct a domain $S$ such that the left part of $S$ is a large rectangle $[0,c] \times [0,1]$, and the right part of $S$ is a copy of $S'$ (see Figure 7.2). Then, because the problem of computing $b$ is equivalent to decide the area of $S'$ (i.e., $area(S') = 2b - c$), the theorem is proved. □

In other words, Theorem 7.2.3 shows that the complexity of computing the bisector at a given angle of a $P$-approximable bounded set with a positive thickness around the bisector is between $P^{\#P}$ and $P^{\#P_1}$ (more precisely, $P^{\#P_1[1]}$, since to compute an approximation to the area of $S'$, the $\#P_1$ oracle will only be queried once; for details, see Chou and Ko [18]). Whether $P^{\#P} = P^{\#P_1}$ is still an open question in discrete complexity theory (see, e.g., Ogihara et al. [56]).

We present a pure mathematical result below that any Jordan domain has a positive thickness around any bisector; furthermore, the thickness is always greater than a positive constant (i.e., bounded below).

**Lemma 7.2.4** *Let $S$ be a Jordan domain and $\Gamma$ the boundary of $S$. Then there exists a real number $\Delta > 0$, such that for any line $L$ that divides $S$ into two parts $S_1$ and $S_2$ with $|area(S_1) - area(S_2)| < area(S)/3$, there is a point $Q$ in the intersection $L \cap S$ of $L$ and $S$ such that* $\mathrm{dist(Q, \Gamma)} > \Delta$. *It follows that $S$ has a positive thickness around all bisectors.*

*Proof.* Let $f : [0, 1] \to \mathbb{R}^2$ be a continuous function such that $f$ is 1-1 on $[0, 1)$, $f(0) = f(1)$ and the image of $f$ is $\Gamma$. Let $\{t\}$ denote the distance of real number $t$ and integer points (e.g., $\{0.6\} = 0.4$, $\{1\} = 0$). Let $K > 0$ be a real number such that $\pi K^2 < area(S)/3$ and $K < area(S)/(6R)$, where $R$ is the radius of a disk that covers $S$. Then there exists a real number $H > 0$ such that for any $t_1, t_2 \in [0, 1]$, $\{t_1 - t_2\} < H \Rightarrow |f(t_1) - f(t_2)| < K$.

Let $A = \{(t_1, t_2) \in [0, 1]^2 : t_1 < t_2, |f(t_1) - f(t_2)| \geq 3K\}$. It is clear that $A$ is closed. From the assumption of $K < area(S)/(6R)$, $A$ is nonempty, for otherwise, all $\Gamma$ is contained in a rectangle of dimensions $3K \times 2R < area(S)$, which is a contradiction. Now we define a function $g$ on $A$ as follows. Let $(t_1, t_2) \in A$. If $\gamma$ is a path from $f(t_1)$ to $f(t_2)$ such that $\gamma \subseteq \overline{S}$, we let $\gamma_K$ denote the portion of $\gamma$ whose distance to $f(t_1)$ and $f(t_2)$ is no less than $K$, i.e., $\gamma_K = \{Q \in \gamma : \mathrm{dist(Q, f(t_i))} \geq \mathrm{K, i} = 1, 2\}$. From the definition of $A$, $\gamma_K$ is nonempty. $g(t_1, t_2)$ is the least upper bound of the distances $\mathrm{dist}(\gamma_\mathrm{K}, \Gamma)$ over all paths $\gamma \subseteq \overline{S}$ from $f(t_1)$ to $f(t_2)$. Roughly speaking, there is a tube that connects $f(t_1)$ and $f(t_2)$ such that the tube is of width$\geq 2g(t_1, t_2)$ in the middle. We have $g(t_1, t_2) > 0$ since there exists a path $\gamma \subseteq \overline{S}$ from $f(t_1)$ to $f(t_2)$ such that $\gamma_K$ is strictly inside $\Gamma$, which implies that $g(t_1, t_2) \geq \mathrm{dist}(\gamma_\mathrm{K}, \Gamma) > 0$.

It is an interesting problem to prove that $g$ is continuous. For any real number $\epsilon > 0$, there exists a real number $\delta > 0$ such that for any $t_1, t_2 \in [0, 1]$, $\{t_1 - t_2\} < \delta \Rightarrow |f(t_1) - f(t_2)| < \epsilon/2$. For any two pairs $(t_1, t_2), (t'_1, t'_2) \in A$ such that $|t_i - t'_i| < \delta$ for $i = 1, 2$, let $\gamma$ be a path from $f(t_1)$ to $f(t_2)$ such that $\mathrm{dist}(\gamma_\mathrm{K}, \Gamma) \geq$

$g(t_1, t_2) - \epsilon/2$. Assume that $t_1 < t_1'$ and $t_2 < t_2'$. Let $\gamma'$ be the path consisting of $\gamma$, $f([t_1, t_1'])$ and $f([t_2, t_2'])$ (removing any repeated portion). Then $g(t_1', t_2') \geq \text{dist}(\gamma_K', \Gamma) \geq \text{dist}(\gamma_K, \Gamma) - \max(|f(t_1) - f(t_1')|, |f(t_2) - f(t_2')|) > (g(t_1, t_2) - \epsilon/2) - \epsilon/2 = g(t_1, t_2) - \epsilon$. Symmetrically, $g(t_1, t_2) > g(t_1', t_2') - \epsilon$. Therefore, $|g(t_1, t_2) - g(t_1', t_2')| < \epsilon$. Thus, $g$ is continuous.

Since $g$ is continuous on a bounded closed set, $g$ assumes its minimum at some points. That is, there exists a real number $\Delta > 0$ such that $g(t_1, t_2) \geq 2\Delta$ for any $(t_1, t_2) \in A$.

Now we prove the lemma. From the assumption of $K$, there exist $t_1, t_2 \in [0, 1]$ such that $f(t_1)$ and $f(t_2)$ are on different sides of $L$ and are the furthest points away from $L$ among all points of $\Gamma$ on two sides of $L$, respectively. Then $\text{dist}(f(t_i), L) > K$ for $i = 1, 2$ and $|f(t_1) - f(t_2)| > 3K$. Assume $t_1 < t_2$, then $(t_1, t_2) \in A$ and $g(t_1, t_2) \geq 2\Delta$. Then there exists a path $\gamma \subseteq \overline{S}$ from $f(t_1)$ to $f(t_2)$ such that $\text{dist}(\gamma_K, \Gamma) \geq g(t_1, t_2) - \Delta \geq 2\Delta - \Delta = \Delta$. The curve $\gamma_K$ must intersect $L$, since $f(t_1)$ and $f(t_2)$ are on different sides of $L$ and $\text{dist}(f(t_i), L) > K$ for $i = 1, 2$. Pick any point $Q$ in $\gamma_K \cap L$, we have $\text{dist}(Q, \Gamma) \geq \Delta$. □

**Remark:** In the proof above, we can show further that $g(t_1, t_2) = \text{dist}(\gamma_K, \Gamma)$ for some path $\gamma \subseteq \overline{S}$ from $f(t_1)$ to $f(t_2)$, but this requires the *Arzela-Ascoli Theorem* in real analysis (see, e.g., Rudin [65]), which complicates the proof unnecessarily.

Note that by Lemma 7.2.4, any Jordan domain has a thickness $W$ around any bisector such that $W$ is greater than a positive constant $2\Delta$. Then we have the following corollary from Theorem 7.2.3.

**Corollary 7.2.5** *In the following,* $(a) \Rightarrow (b) \Rightarrow (c)$*:*

$(a)$ $FP = \#P$.

$(b)$ *For any $P$-approximable Jordan domain $S$, the bisector of $S$ at any polynomial-time computable angle $\alpha \in [0, \pi)$ is polynomial-time computable.*[1]

[1]Note that the bisector is defined by $y = \tan(\alpha)x + b$ if $\alpha \neq \pi/2$ and $x = b$ if $\alpha = \pi/2$, for some

(c) $FP_1 = \#P_1$.

There is also another version of bisecting a single set $S$: the bisector is through a fixed point $Q$, instead of forming a fixed angle with the positive $x$-axis. If the fixed point $Q$ is away from the set $S$ in the sense that $Q$ and $S$ are on different sides of a line (i.e., $Q$ and $S$ are *linearly separable*), and $S$ has a positive thickness around a bisector $L$ of $S$ that goes through $Q$, then $L$ is the unique such bisector. Note that here thickness around a line $L$ is more naturally defined as the rate of area change over angle change. More precisely, suppose $Q$ and $S$ are separated by a horizontal line and $Q$ is below $S$. For an angle $\alpha \in [0, \pi)$, let $L_\alpha$ be the half line through $Q$ at angle $\alpha$, and $S_{angle<\alpha}$ be the part of $S$ on the right of $L_\alpha$. Now the thickness of $S$ at line $L_\alpha$ (or we may say *the thickness of $S$ at angle $\alpha$ with respect to $Q$*) is defined as

$$\liminf_{\delta \to 0} \frac{area(S_{angle<\alpha+\delta}) - area(S_{angle<\alpha})}{\delta},$$

and similarly we can define positive thickness around a line (or angle). We remark that this definition is consistent with the previous one in the sense that if $S$ has a positive thickness around a line $L_\alpha$ by the previous definition, Definition 7.2.1, then by the current definition $S$ still has a positive thickness around angle $\alpha$ with respect to a point $Q$ on $L_\alpha$ that is linearly separable from $S$. (However, the exact thickness values under these two definitions are not the same.)

**Theorem 7.2.6** *In the following, (a) $\Rightarrow$ (b) $\Rightarrow$ (c):*

(a) $FP = \#P$.

(b) *Let $S$ be a $P$-approximable set such that the origin $O$ and $S$ are linearly separated by a horizontal line with $O$ below $S$. If a line $L_\alpha$ ($\alpha \in [0, \pi)$) through $O$ is a bisector of $S$ and $S$ has a positive thickness around angle $\alpha$ with respect to $O$, then $L_\alpha$ is the unique such bisector of $S$ and $\alpha$ is polynomial-time computable.*

(c) $FP_1 = \#P_1$.

real number $b$, and the problem is to compute $b$ in polynomial time.

*Proof.* The proof is similar to that of Theorem 7.2.3 and thus is omitted. Note that for the part $(b) \Rightarrow (c)$, we still use the domain $S$ shown in Figure 7.2, and we can make the bisector $L_\alpha$ only intersect the left part of $S$, i.e., the rectangle, then the problem of finding $L_\alpha$ is equivalent to decide the area of $S$. □

On the other hand, if the fixed point $Q$ can be inside $S$, there is no fixed complexity bound for finding the bisector of $S$ through $Q$.

**Theorem 7.2.7** *Suppose $\alpha \in [0, \pi)$ is a computable number. Then there exists a P-approximable/recognizable convex Jordan domain $S$, such that the unique bisector of $S$ through the origin $O$ is at angle $\alpha$.*

*Proof.* The idea of the construction of $S$ is similar to that in the proof of Theorem 7.2.2. The boundary of $S$ is the image of a function $f : [0, 2\pi] \to \mathbb{R}^2$ such that for any $\beta \in [0, 2\pi]$, $f(\beta) = g(\beta)\langle\cos(\beta), \sin(\beta)\rangle$, where $g : [0, 2\pi] \to \mathbb{R}$ is a function whose values are always positive. It suffices to just describe the boundary of $S$ in order to explain what $f$ is.

Without loss of generality, we assume that $\alpha \in (1/4, 1/2)$. We follow the notations of the proof of Theorem 7.2.2, and define two sequences $\{\underline{d}_n\}$ and $\{\overline{d}_n\}$ that binary converge to $\alpha$ from below and upper, function $s(n)$, sets $I_1$ and $I_2$, the intervals of the first and second kinds, and let $a_{m,n}$ be the $n$-th smallest number in $I_m$, where $m \in \{1, 2\}$.

For any $n \in \mathbb{N}$, let $i = a_{1,n}$ and $j = a_{1,n+1}$ and $h = \max(a_{1,n+1}, a_{2,n+1})$. We have $\underline{d}_j - \underline{d}_i \geq 2^{-s(h)}$. Then on the $n$-th interval $[\underline{d}_i, \underline{d}_j]$ of the first kind, the image of $f$ is identical to the part of the unit circle at the sector between angles $\underline{d}_i$ and $\underline{d}_j$, except that there is a bump on $[\underline{d}_i, \underline{d}_i + 2^{-s(h)}]$; more precisely, the image of $f$ on $[\underline{d}_i, \underline{d}_i + 2^{-s(h)}]$ is the union of two tangents: that is, on $[\underline{d}_i, \underline{d}_i + 2^{-s(h)}]$, the image of $f$ is a piecewise linear curve with breakpoints $\langle\cos(\underline{d}_i), \sin(\underline{d}_i)\rangle$, $\langle\cos(\underline{d}_i + 2^{-(s(h)+1)}), \sin(\underline{d}_i + 2^{-(s(h)+1)})\rangle / \cos(2^{-(s(k)+1)})\rangle$ and $\langle\cos(\underline{d}_i + 2^{-s(h)}), \sin(\underline{d}_i + 2^{-s(h)})\rangle$. Similarly, we define

$f$ on the intervals of the second kind. The image of $f$ on $[0, 2\pi] - [1/4, 1/2]$ is identical to the unit circle on the same domain.

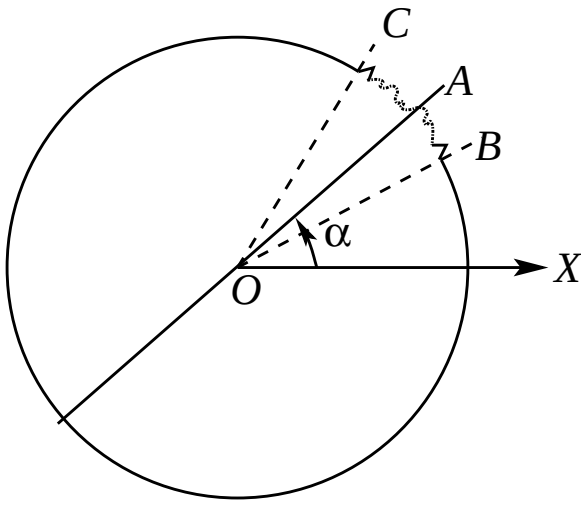


**Figure 7.3:** The domain $S$ for (b) Theorem 7.2.7.

Figure 7.3 illustrates the domain $S$: the lines $OA$, $OB$ and $OC$ are at angles $\alpha$, 1/2, 1/4 respectively; the bumps on different sides of $OA$ can be paired up such that the two bumps in each pair are of the same area. We can check that $S$ is a $P$-approximable/recognizable Jordan domain similar to the proof of Theorem 7.2.2. Also, $S$ is convex, since its boundary $\partial S$ consists of circular arcs and tangents. (The domain $S$ does not look convex in the figure because we want to emphasize the bumps in a small region.) As shown in the figure, the area of bumps in the sector $AOB$ is the same as the area of bumps in the sector $AOC$, so $OA$ is a bisector of $S$ that goes through the origin $O$. There are no other such bisectors because for any line $OD$ other than $OA$, on one side of $OD$, there are only a finite number of bumps, while on the other side, there are infinitely many bumps. □

## 7.3 Bisecting Two Sets Simultaneously

Mathematically, the Pancake Theorem follows the *intermediate value theorem* and the *Borsuk-Ulam Theorem* (see, e.g., Fulton [30]). For completeness, we show how the

intermediate value theorem implies the Borsuk-Ulam Theorem and how the Borsuk-Ulam Theorem implies the Pancake Theorem on the two-dimensional plane (see, e.g., Fulton [30]).

Let $C$ be the unit circle with its center at the origin. For $\alpha \in [0, 2\pi]$, we use $\mathbf{z}_\alpha$ to denote a point $\langle \cos\alpha, \sin\alpha \rangle$. The Borsuk-Ulam Theorem on the two-dimensional plane states that any continuous function $f : C \to \mathbb{R}$ must map a pair of antipodal points $\mathbf{z}_{\alpha_0}$ and $\mathbf{z}_{\alpha_0+\pi}$ to the same value. To see this, let $g : C \to \mathbb{R}$ be the function defined by $g(\mathbf{z}_\alpha) = f(\mathbf{z}_\alpha) - f(\mathbf{z}_{\alpha+\pi})$. Then $g$ is a continuous function satisfying $g(\mathbf{z}_0) = -g(\mathbf{z}_\pi)$. From the intermediate value theorem, there must exists a number $\alpha_0 \in [0, \pi]$ such that $g(\mathbf{z}_{\alpha_0}) = 0$, i.e., $f(\mathbf{z}_{\alpha_0}) = f(\mathbf{z}_{\alpha_0+\pi})$.

Now we show how the Borsuk-Ulam Theorem implies the Pancake Theorem. For simplicity, we only consider two Jordan domains $S$ and $S'$. We assume that $S$ and $S'$ are both inside the unit circle. Now we define a function $f : C \to \mathbb{R}$ as follows: for any $\mathbf{z}_\alpha \in C$, let $L_{\alpha+\pi/2}$ be the bisector of $S$ at angle $\alpha + \pi/2$, which divides $S'$ into two parts $S'_1$ and $S'_2$ with $S'_1$ closer to $\mathbf{z}_\alpha$, and $f(\mathbf{z}_\alpha)$ is the area of $S'_1$. The function $f$ is continuous because, when $\mathbf{z}_\alpha$ and $\mathbf{z}_{\alpha'}$ are close to each other, inside the unit circle the bisectors $L_{\alpha+\pi/2}$ and $L_{\alpha'+\pi/2}$ should be close to each other too, since $S$ has a positive thickness $\geq 2\Delta$ around any bisector (Lemma 7.2.4). Then from the Borsuk-Ulam theorem, there exists a number $\alpha_0 \in [0, \pi]$ such that $f(\mathbf{z}_{\alpha_0}) = f(\mathbf{z}_{\alpha_0+\pi})$, which implies that $L_{\alpha_0+\pi/2}$ bisects $S'$ too.

Next we present some results on the computability and complexity of the intermediate value theorem and the Borsuk-Ulam Theorem.

**Theorem 7.3.1** *(a) Let $f : [0, 1] \to \mathbb{R}$ be a computable function such that $f(0)f(1) < 0$. Then there exists a computable number $r \in (0, 1)$ such that $f(r) = 0$.*

*(b) (Ko [40, Theorem 4.4]) For any computable number $r \in (0, 1)$, there exists a polynomial-time computable function $f : [0, 1] \to \mathbb{R}$ such that (1) $f$ is strictly*

*increasing;* (2) $f(0)f(1) < 0$*; and* (3) $f(r) = 0$.

Part (a) of Theorem 7.3.1 provides a positive result that every computable function that changes sign has a computable zero, while part (b) shows a negative result that it is arbitrarily hard to compute it even if there is only one zero and the function is polynomial-time computable. As pointed out in Ko [40], if the function $f$ has a polynomial inverse modulus, the unique zero of $f$ is polynomial-time computable.

In the following, we prove a negative result about the complexity of the Borsuk-Ulam Theorem.

**Theorem 7.3.2** *Let $\alpha \in [0, \pi)$ be a computable real number. Then there exists a polynomial-time computable function $f : C \to \mathbb{R}$ such that $(\mathbf{z}_\alpha, \mathbf{z}_{\alpha+\pi})$ is the only pair of antipodal points that satisfies $f(\mathbf{z}_\alpha) = f(\mathbf{z}_{\alpha+\pi})$.*

*Proof.*Proof (Ideas) Without loss of generality, suppose $\alpha \in (\pi/3, 2\pi/3)$. We can construct a function $g : [0, 2\pi] \to \mathbb{R}$ such that

(1) $g$ is strictly increasing on $[0, \pi]$ with $g(\alpha) = 1$.

(2) $g$ is piecewise-linear on $[\pi, 2\pi]$ with $g(4\pi/3) = 1$, $g(5\pi/3) = 1$ and $g(2\pi) = g(0)$.

The construction of (1) follows that of part (b) of Theorem 7.3.1. It is obvious that $g(\alpha) = g(\alpha + \pi)(= 1)$. To prove that $(\alpha, \alpha + \pi)$ is the only such pair, we note that for $\beta \in [0, \pi/3]$, $g(\beta) < 1$ and $g(\beta + \pi) \geq 1$; for $\beta \in [2\pi/3, \pi]$, $g(\beta) > 1$ and $g(\beta + \pi) \leq 1$; for $\beta \in [\pi/3, 2\pi/3]$, $g(\beta + \pi) = 1$ but $g(\beta) \neq 1$ if $\beta \neq \alpha$.

Let $f(\mathbf{z}_\beta) = g(\beta)$ for any $\beta \in [0, 2\pi]$. From the above discussion, it is clear that $(\mathbf{z}_\alpha, \mathbf{z}_{\alpha+\pi})$ is the only pair of antipodal points that satisfies $f(\mathbf{z}_\alpha) = f(\mathbf{z}_{\alpha+\pi})$. □

Now we consider the Pancake Theorem. Again the computability problem has an affirmative answer and we omit its proof. Our results below show that in general it is arbitrarily hard to compute the common bisector of two $P$-approximable sets even if (1) each of these sets has a positive thickness around any bisector; and (2) there

is exactly one common bisector. However, if the two sets are linearly separable, the complexity of the problem is characterized by counting classes $\#P$ and $\#P_1$.

Note that we only need to compute the angle $\alpha$ from the positive $x$-axis to the common bisector $L$, since once $\alpha$ is known, the common bisector can be computed.

**Theorem 7.3.3** *Let $\alpha \in [0, \pi)$ be a computable number. Then there exist two convex $P$-approximable/recognizable Jordan domains $S$ and $S'$ that have only one common bisector $L$, and the angle from the positive $x$-axis to $L$ is $\alpha$.*

*Proof.* The domain $S$ is the same as in Theorem 7.2.7, and $S'$ is the unit disk. □

There is another version of the Pancake theorem, which states that for any bounded set $S \subseteq \mathbb{R}^2$, there exist two lines that are perpendicular to each other and divide $S$ into four parts of equal area.

**Theorem 7.3.4** *Let $\alpha \in [0, \pi)$ be a computable number. Then there exists a convex $P$-approximable/recognizable Jordan domain $S$ whose boundary is polynomial-time computable, such that the following properties are satisfied:* (1) *there exist exactly two lines $L_1$ and $L_2$ that are perpendicular to each other and divide $S$ into four parts of equal area;* (2) *the angle from the positive $x$-axis to $L_1$ is $\alpha$; and* (3) *both lines $L_1$ and $L_2$ pass through the origin.*

**Proof.** The domain $S$ is similar to the one constructed in the proof of Theorem 7.2.7, except that we remove some parts from $S$ in intervals $[\alpha + 3\pi/2, 2\alpha + 3\pi/2 - 1/4]$ and $[2\alpha + \pi/2 - 1/2, \alpha + \pi/2]$, with the area in each removed part equals what is added to $S$ (compared to the unit disk) in the interval $[1/4, \alpha]$ (and in the interval $[\alpha, 1/2]$). Then the area of $S$ is the same as the unit disk, that is, $area(S) = \pi$. See Figure 7.4(a) for an illustration of $S$. It is easy to check that the two lines $L_1$ and $L_2$ satisfying conditions (2) and (3) divide $S$ into four parts of equal area.

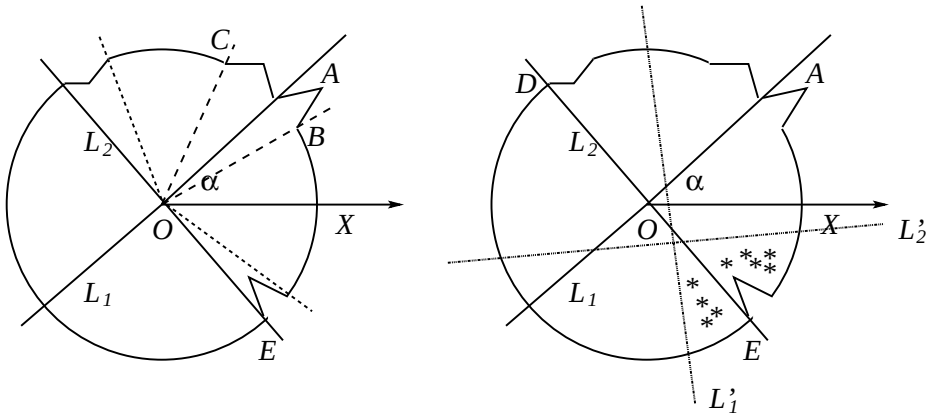


**Figure 7.4:** (a) The domain $S$ for Theorem 7.3.4, (b) Divisions with two lines.

It remains to show that $L_1$ and $L_2$ are the only pair of lines that can do so. Suppose that two lines $L_1'$ and $L_2'$ are perpendicular to each other and divide $S$ into four parts of equal area $area(S)/4 = \pi/4$, as in Figure 7.4(b). There are two cases.

**Case (a):** Both $L_1'$ and $L_2'$ contain the origin $O$. It is easy to check that $\{L_1, L_2\} = \{L_1', L_2'\}$.

**Case (b):** Not the above case. Without loss of generality, assume that $L_1'$ does not contain the origin $O$. $L_1'$ must intersect the arc from $D$ through $A$ to $E$ of the boundary of $S$, for otherwise $L_1'$ divides $S$ into two parts, with the area of one of them less than $area(S)/2 = \pi/2$, since it is smaller than a half unit disk. Without loss of generality, assume that $L_1'$ intersects the arc from $D$ to $A$ (see Figure 7.4(b)). Then $D$ and $O$ are on the same side of $L_1'$, for otherwise $L_1'$ divides $S$ into two parts, with the area of the part containing $D$ less than $area(S)/2$. Then $D$ and $O$ must be on the same side of $L_2'$, for otherwise the part of $S$ divided by $L_1'$ and $L_2'$ that contains $O$ is larger than $1/4$ of a unit disk. Now the part of $S$ divided by $L_1'$ and $L_2'$ that contains $E$ (i.e., the part marked with $*$'s in Figure 7.4(b)) is of area $< area(S)/4$, which is a contradiction. $\square$

Recall that we say two sets $S$ and $S'$ are linearly separable if they are on different

sides of some line. If the two sets $S$ and $S'$ are linearly separable and each of them has a positive thickness around all its bisectors, there exists exactly one common bisector of $S$ and $S'$.

**Theorem 7.3.5** *In the following,* $(a) \Rightarrow (b) \Rightarrow (c)$*:*

$(a)$ $FP = \#P$.

$(b)$ *For any two linearly separable bounded* $P$*-approximable sets* $S$ *and* $S'$ *with positive thickness around all bisectors, the unique line* $L$ *that bisects simultaneously the two sets is polynomial-time computable.*

$(c)$ $FP_1 = \#P_1$.

*Proof.* The spirits of the proof are quite similar to the proof of Theorem 7.2.3.

For the proof of (a)⇒ (b), suppose that $S$ and $S'$ are separated by a vertical line $L$(see Figure 7.5). For any line $L'$ that intersects $L$, we define its direction as from a point on $L'$ lying on the left of $L$ to a point on $L'$ lying on the right of $L$. Then we can say the angles of such lines with respect to the positive $x$-axis are from $-\pi/2$ to $\pi/2$.

We define a function $g : (-\pi/2, \pi/2) \to \mathbb{R}$ as follows: for any $\alpha \in (-\pi/2, \pi/2)$, there exists uniquely a line $L_\alpha$ that bisects $S$ at angle $\alpha$. $L_\alpha$ will divide $S'$ into two parts $S_1$ and $S_2$, with $S_1$ on the right of $L_\alpha$ (recall that $L_\alpha$ has a direction). Let $g(\alpha) = area(S_1) - area(S_2)$. Function $g$ has only one root $\alpha_0$, with $L_{\alpha_0}$ being the unique common bisector of $S$ and $S'$. Next we will show that under condition (a), $\alpha_0$ is polynomial-time computable, which further implies that $L_{\alpha_0}$ is polynomial-time computable by Theorem 7.2.3.

First $g(\alpha)$ has a linear inverse modulus when $\alpha$ is close to the root $\alpha_0$ of $g$: as in Figure 7.5, for two angles $\alpha, \beta \in (-\pi/2, \pi/2)$ with $\alpha < \beta$, $S'$ and the intersection of $L_\alpha$ and $L_\beta$ must be on different sides of $L$, which implies that $g(\alpha) < g(\beta)$; more precisely, $g(\beta) - g(\alpha)$ is twice the area of the intersection of $S'$ and a sector region of

angle $\beta - \alpha$, and from the thickness condition of $S'$, $g(\beta) - g(\alpha) > m(\beta - \alpha)$ for some $m > 0$, provided that $\alpha$ and $\beta$ are sufficiently close to $\alpha_0$. Therefore, by Corollary 4.7 of Ko [40], the root $\alpha_0$ of $g$ is polynomial-time computable if $g$ is polynomial-time computable. According to Theorem 7.2.3, $g$ is polynomial-time computable under condition (a), thus the proof is completed.

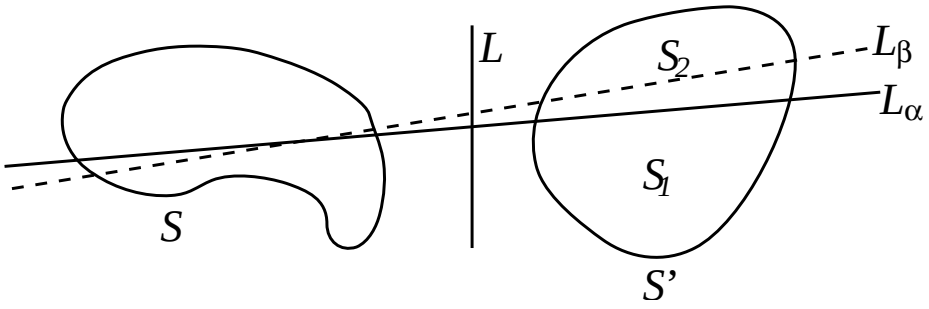


**Figure 7.5:** The domains $S$ and $S'$ for (a)⇒ (b) of Theorem 7.3.5.

For the proof of (b)⇒ (c), we let $S$ be a domain similar to the one in the proof of (b)⇒(c) of Theorem 7.2.3 and $S'$ a disk lying below $S$. Then the common bisector of $S$ and $S'$ must go through the center of $S'$. The proof is completed by following $(b) \Rightarrow (c)$ of Theorem 7.2.6. □

# Chapter 8

# Conclusion and Open Problems

In this thesis we have studied the computational complexity of a few fundamental problems in continuous mathematics. These results show that there are natural problems in continuous mathematics that are characterized by standard complexity classes of discrete complexity theory. We have gained some insights into the nature of continuous complexity theory and into the relationship between continuous complexity theory and discrete complexity theory.

**The logarithm and square root problems and their generalization: analytic continuation.** We have shown that the logarithm and square root problems have an upper bound $\#P$. In addition, we showed tight lower bounds $\#P$ and $\oplus P$ for them, respectively. Both of these problems are special examples of the more general analytic continuation problem, which can be stated as follows.

> Let $S$ be a bounded, simply connected domain with a polynomial-time computable boundary $\partial S$. Suppose $\mathbf{z}_0$ is a fixed point in $S$ and $f : S \to \mathbb{C}$ is an analytic function such that $f$ is polynomial-time computable in a neighborhood of $\mathbf{z}_0$. Compute $f(\mathbf{z})$ for any point $\mathbf{z} \in S$.

We point out that, for general analytic continuation problems, the integration method may not work. Recall that we achieved the upper bound $P^{\#P}$ using the method of integration along the curve $\partial S$. We can do this because the derivative of $\log(\mathbf{z}-\mathbf{a})$ with respect to $\mathbf{z}$ is simply $1/(\mathbf{z}-\mathbf{a})$, which is independent of the shape of $S$ and also enables the trick of cancellation to work. In general, however, the complexity of computing the derivative $f'$ of $f$ might be as high as computing $f$ itself; in addition, $f'$ is not necessarily of a simple form, and the integral of $f'$ on the boundary curve may not exist and, even if it exists, the complexity of computing the integral may be higher. One possible solution to this is to consider analytic continuation on a path that lies entirely inside the domain $S$. Chou and Ko [19; 21] have investigated the complexity issues of finding a path inside a domain. A recent study of Ko and Yu [46] provided an upper bound of exponential space if the boundary of the domain has a polynomial inverse modulus of continuity.

Recall that we say a function $f : [0,1] \to \mathbb{R}^2$ represents a Jordan curve if (i) $f$ is one-to-one on $[0,1)$ and $f(0) = f(1)$, and (ii) the image of $[0,1]$ under $f$ is the curve $\Gamma$. Note that if $f$ represents a Jordan curve and $f$ is polynomial-time computable, then $f$ has a polynomial modulus of continuity; that is, there is a polynomial $p$ such that, for all $n > 0$, $\delta(s,t) \leq 2^{-p(n)}$ implies $f(s,t) \leq 2^{-n}$.

We say a function $m : \mathbb{N} \to \mathbb{N}$ is an *inverse modulus of continuity* of a function $f : [0,1] \to \mathbb{R}^2$ if there is a polynomial function $p : \mathbb{N} \to \mathbb{N}$ and an integer $n_0$ such that the following holds for all $n > n_0$: For any two points $s, t \in [0,1]$ $|f(s)-f(t)| > 2^{-p(n)}$ whenever $\delta(s,t) > 2^{-n}$. We say a Jordan curve $\Gamma$ is polynomial-time computable and has a polynomial inverse modulus of continuity if there is a polynomial-time computable function $f$ representing $\Gamma$ which has a polynomial inverse modulus of continuity.

Jordan curves that are polynomial-time computable and have polynomial inverse moduli of continuity are quite common, and even include some fractal curves(see

Ko [41] and Ko and Weihrauch [43]). With this extra condition, the upper bounds of quite a few problems, including the membership problem and the path problem, are reduced. For details, see [46].

Here we only point out, with the polynomial inverse modulus condition, we can obtain an upper bound of exponential space for the general analytic continuation problem.

**Theorem 8.0.6** *Assume that $S$ is a simply connected domain on the two-dimensional plane whose boundary $\Gamma$ is represented by a polynomial-time computable function which has a polynomial inverse modulus of continuity, and that $\mathbf{z}_0$ is a point in $\mathrm{Int}(\Gamma)$ with $\mathrm{dist}(\mathbf{z}_0, \Gamma) = 2^{-\mathrm{n}_0}$. Also assume that function $g$ is analytic on $S$, and that the power series of $g$ at $\mathbf{z}_0 \in \mathrm{Int}(\Gamma)$ is polynomial-time computable. Then, there exists a polynomial $q(n)$ such that for any integer $n \geq n_0$ and any point $\mathbf{z} \in \mathrm{Int}(\Gamma)$ with $\mathrm{dist}(\mathbf{z}, \Gamma) \geq 2^{-\mathrm{n}}$, the power series of $G$ at $\mathbf{z}$ is computable using at most $2^{q(n+k)}$ cells.*

This provides us with an EXPSPACE upper bound for the general analytic continuation problem, when we add the polynomial inverse modulus of continuity condition to the boundary $\partial S$ of $S$. Whether the gap between this upper bound and the lower bound $\#P$ can be further narrowed, possibly with additional constraints on the domain $S$ is an interesting open question.

***NC* and Log-space analytic functions.** We studied some fundamental problems in parallel time complexity theory of continuous mathematics. We showed that the common operators, such as integration and differentiation, on $NC$ and Log-space functions are not necessarily closed in $NC$ or Log-space, respectively, unless $P$ collapses to $NC$ or Log-space, respectively. On the other hand, analyticity helps, just like in the polynomial-time complexity theory, to reduce the complexity and keep it closed in $NC$ or Log-space for these operators. The complexity of finding all

zeros of an analytic function inside a Jordan curve is also proved to be closed in $NC$. However, the question of whether it is closed in Log-space (that is, if all settings are in Log-space, whether the complexity of finding these zeros is still Log-space) is still open.

Related to this question, the representations of real numbers in complexity classes $NC$ and Log-space do not relate to each other the same way as those in the class $P$. It is worth further study to understand the exact effect of these somewhat surprising relations, and to find out which representations are more suitable for $NC$ and Log-space functions and operators.

**Topology and geometry problems.** In Chapters 5 to 7, we studied the complexity issues of some geometry problems. These include the convex hull problem, the circumscribed rectangle problem and the pancake problem. By studying them in the computable analysis setting, we extended the scope of the objects from polygons or finite sets of points to polynomial-time computable Jordan domains. Our results complement the results in computational geometry about these problems, and provide different insights into the inherent difficulties of these problems. Namely, we showed that the convex hull problem, which has a number of simple algorithms in the computational geometry setting, is as hard to solve as any discrete $NP$-complete problem. While the circumscribed rectangle problem and the pancake problem both have linear time algorithm in the computational geometry setting, the algorithms are considerably more complicated. We showed, in the computable analysis setting, that they have higher complexity than the convex hull problem: The circumscribed rectangle problem has complexity $\Sigma_2^P$, the second level of the polynomial-time hierarchy, and the pancake problem has even higher complexity of $\#P$, essentially requiring the computation of integrals.

There are a number of open questions concerning these problems. First, for the

convex hull problem, we showed that if a domain has a $P$-computable, finite-length Jordan curve as the boundary, then the area of its convex hull must be a $\#NP$ real number. However, the exact complexity of the class $\#NP_R$ of $\#NP$ real numbers, particularly its relation with $\#P_R$, the class of $\#P$ real numbers, is not clear, and should be clarified.

For the circumscribed rectangle and square problem, we would like to point out a related, and very interesting question of finding the *inscribed square* problem: does any Jordan curve contain four vertices of a square? Mathematicians still do not know whether the answer is affirmative or not, although it is for sufficiently smooth Jordan curve [69]. Some interesting questions would arise: what about the computability and complexity of the inscribed square problem provided that there exists one inscribed square for the given Jordan curve? Are there (computable) inscribed squares for computable Jordan curves?

Finally, for the pancake problem, we have used a reasonable thickness condition to characterize the complexity of bisecting a set in the given direction. It would be interesting to find out whether there are other natural conditions on the domains under which the complexity $\#P$ can be reduced.

# References


[1] T. Abbott, E. D. Demaine, M. L. Demaine, D. Kane, S. Langerman, J. Nelson, and V. Yeung. Dynamic ham-sandwich cuts of convex polygons in the plane. In *CCCG*, pages 61–64, 2005.

[2] O. Aberth. Analysis in the computable number field. *Journal of the Association for Computing Machinery*, 15:275–299, 1968.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, Reading, 1974.

[4] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theor. Comput. Sci.*, 107(1):3–30, 1993.

[5] K. Ambos-Spies, K. Weihrauch, and X. Zheng. Weakly computable real numbers. *Journal of Complexity*, 16(4):676–690, 2000.

[6] D. Bailey, P. Borwein, and S. Plouffe. On the rapid computation of various polylogarithmic constants. *Mathematics of Computation*, 66(218):903–913, 1997.

[7] P. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM J. Comput.*, 15(4):994–1003, 1986.

[8] M. D. Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, New York, 2nd edition, 2000.

[9] M. Bläser. Uniform computational complexity of the derivatives of $C^\infty$-functions. *Theor. Comput. Sci.*, 284(2):199–206, 2002.

[10] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[11] D. P. Bovet and P. Crescenzi. *Introduction to the theory of complexity*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1994.

[12] M. Braverman and S. Cook. Computing over the reals: Foundations for scientific computing. *Notices of the AMS*, 53(3), March 2006.

[13] R. L. Burden and J. D. Faires. *Numerical Analysis*. Brooks/Cole, seventh edition, 2001.

[14] P. B'urgisser, M. Clausen, M. A. Shokrollahi, and T. Lickteig. *Algebraic Complexity Theory*. Springer-Verlag, Berlin, Germany, 1997.

[15] M. P. Carpentier and A. F. D. Santos. Solution of equations involving analytic functions. *J. Comput. Phys.*, 45:210–220, 1982.

[16] G. Ceitin. Algorithmic operators in constructive metric spaces. *Translations of the AMS*, 64:1–80, 1967.

[17] A. Chiu, G. I. Davida, and B. E. Litow. Division in logspace-uniform $NC^1$. *ITA*, 35(3):259–275, 2001.

[18] A. Chou and K.-I. Ko. Computational complexity of two-dimensional regions. *SIAM Journal on Computing*, 24:923–947, 1995.

[19] A. W. Chou and K.-I. Ko. On the complexity of finding paths in a two-dimensional domain I: Shortest paths. *Mathematical Logic Quarterly*, 50(6):551–572, 2004.

[20] A. W. Chou and K.-I. Ko. The computational complexity of distance functions of two-dimensional domains. *Theor. Comput. Sci.*, 337(1-3):360–369, 2005.

[21] A. W. Chou and K.-I. Ko. On the complexity of finding paths in a two-dimensional domain II: Picewise straight-line paths. In V. Brattka, L. Staiger, and K. Weihrauch, editors, *Proceedings of the 6th Workshop on Computability and Complexity in Analysis*, volume 120 of *Electronic Notes in Theoretical Computer Science*, pages 45–57, Amsterdam, 2005. Elsevier. 6th International Workshop, CCA 2004, Wittenberg, Germany, August 16–20, 2004.

[22] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[23] L. Delves and J. Lyness. A numerical method for locating the zeros of an analytic function. *Math. Comput.*, 1967.

[24] N. A. A. DePano. Rotating calipers revisited: optimal polygonal enclosures in optimal time. In *Proc. ACM South Central Regional Conference*, 1987.

[25] D. Du and K.-I. Ko. Computational complexity of integration and differentiation of convex functions. *System Sci. and Math. Sci.*, 2:70–79, 1989.

[26] D.-Z. Du and K.-I. Ko. *Theory of Computational Complexity*. John Wiley & Sons, New York, 2000.

[27] H. Edelsbrunner and R. Waupotitsch. Computing a ham-sandwich cut in two dimensions. *J. Symb. Comput.*, 2(2):171–178, 1986.

[28] H. Freeman and R. Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Commun. ACM*, 18(7):409–413, 1975.

[29] H. Friedman. On the computational complexity of maximization and integration. *Adv. in Math.*, 53:80–98, 1984.

[30] W. Fulton. *Algebraic Topology: A First Course*. Number 153 in Graduate Texts in Mathematics. Springer, 1997.

[31] R. L. Goodstein. *Recersiee Analysis*. North Holland, 1961.

[32] X. Gourdon and P. Sebah. The Euler Constant: $\gamma$. In *http://numbers.computation.free.fr/Constants/Gamma /gamma.pdf*.

[33] A. Grzegorczyk. Computable funtionals. *Fundamenta Mathematica*, 42:186–202, 1955.

[34] L. Hemaspaandra and H. Vollmer. The satanic notations: Counting classes beyond $\#p$ and other definitional adventures. *SIGACT News*, 26:2–13, 1995.

[35] P. Henrici. *Applied and Computational Complex Analysis*, volume 1-3. John Wiley & Sons, New York, 1974.

[36] H. J. Hoover. Feasible real functions and arithmetic circuits. *SIAM J. Comput.*, 19(1):182–204, 1990.

[37] H. J. Hoover. Real functions, contraction mappings, and P-completeness. *Inf. Comput.*, 93(2):333–349, 1991.

[38] J. E. Hopcroft and J. D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[39] M. E. Houle and G. T. Toussaint. Computing the width of a set. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(5):761–765, 1988.

[40] K.-I. Ko. *Complexity Theory of Real Functions*. Birkhäuser, Boston, 1991.

[41] K.-I. Ko. Recent progress on complexity theory of real functions. In K.-I. Ko and K. Weihrauch, editors, *Computability and Complexity in Analysis*, volume 190 of *Informatik Berichte*, pages 83–90. FernUniversität Hagen, September 1995. CCA Workshop, Hagen, August 19–20, 1995.

[42] K.-I. Ko and H. Friedman. Computational Complexity of Real Functions. *Theoretic Computer Science*, 1982.

[43] K.-I. Ko and K. Weihrauch. On the measure of two-dimensional regions with polynomial-time computable boundaries. In S. Homer and J.-Y. Cai, editors, *Eleventh Annual IEEE Conference on Computational Complexity*, pages 150–159, Los Alamitos, 1996. IEEE Computer Society Press.

[44] K.-I. Ko and F. Yu. Some Computability Problems on Jordan Curves. Technical report, Department of Computer Science, Stony Brook University, NY, 2004.

[45] K.-I. Ko and F. Yu. On the complexity of computing the logarithm and square root functions on a complex domain. *J. of Complexity*, 23(1):2–24, Feburary 2007.

[46] K.-I. Ko and F. Yu. Jordan curves with polynomial inverse moduli of continuity. *Theoretical Computer Science*, (to appear).

[47] K.-I. Ko and F. Yu. On the complexity of convex hulls of subsets of the two-dimensional plane. *Electronic Notes in Theoretical Computer Science*, (to appear).

[48] P. Kravanja and M. V. Barel. *Computing the zeros of Analytic Functions*. Springer-Verlag, 2000.

[49] D. Lacombe. Classes récursivement fermés et fonctions majorantes. *Comptes Rendus Académie des Sciences Paris*, 240:716–718, June 1955. Théorie des fonctions.

[50] C.-Y. Lo, J. Matousek, and W. L. Steiger. Ham-sandwich cuts in $R^d$. In *STOC*, pages 539–545. ACM, 1992.

[51] C.-Y. Lo, J. Matousek, and W. L. Steiger. Algorithms for ham-sandwich cuts. *Discrete & Computational Geometry*, 11:433–452, 1994.

[52] M. H. Meylan and L. Gross. A parallel algorithm to find the zeros of a complex analytic function. *ANZIAM J.*, 44(E):E236–E254, feb 2003.

[53] N. T. Müller. Uniform computational complexity of taylor series. In T. Ottmann, editor, *ICALP*, volume 267 of *Lecture Notes in Computer Science*, pages 435–444. Springer, 1987.

[54] C. A. Neff. Specified precision polynomial root isolation is in NC. *J. Comput. Syst. Sci.*, 48(3):429–463, 1994.

[55] E. Novak. The real number model in numerical analysis. *J. Complex.*, 11(1):57–73, 1995.

[56] M. Ogihara, T. Thierauf, S. Toda, and O. Watanabe. On closure properties of #P in the context of PF #P. *J. Comput. Syst. Sci.*, 53(2):171–179, 1996.

[57] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, Cambridge, UK, 1998.

[58] C. H. Papadimitriou. On graph-theoretic lemmata and complexity classes (extended abstract). In *FOCS*, volume II, pages 794–801. IEEE, 1990.

[59] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, Boston, MA, USA, 1993.

[60] M. S. Petkovic. On initial conditions for the convergence of simultaneous root finding methods. *Computing*, 57(2):163–178, 1996.

[61] M. S. Petkovic, C. Carstensen, and M. Trajkovc. Weierstrass formula and zero-finding methods, 1995.

[62] H. Pirzadeh. Computational geometry with the rotating calipers. Master's thesis, McGill University, 1999.

[63] M. B. Pour-El and J. I. Richards. *Computability in Analysis and Physics*. Perspectives in Mathematical Logic. Springer, Berlin, 1989.

[64] R. Rettinger, X. Zheng, R. Gengler, and B. von Braunmühl. Monotonically computable real numbers. *Mathematical Logic Quarterly*, 48(3):459–479, 2002.

[65] W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, 3rd edition, 1976.

[66] P. J. Schneider and D. Eberly. *Geometric Tools for Computer Graphics*. Elsevier, 2003.

[67] E. Specker. Der Satz vom Maximum in der rekursiven Analysis. In A. Heyting, editor, *Constructivity in mathematics*, Studies in Logic and the Foundations of Mathematics, pages 254–265, Amsterdam, 1959. North-Holland. Proc. Colloq., Amsterdam, Aug. 26–31, 1957.

[68] I. Stojmenovic. Bisections and ham-sandwich cuts of convex polygons and polyhedra. *Inf. Process. Lett.*, 38(1):15–21, 1991.

[69] W. Stromquist. Inscribed squares and square-like quadrilaterals in closed curves. *Mathematika*, 36.

[70] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20:865–877, 1991.

[71] S. Toda and O. Watanabe. Polynomial-time 1-turing reductions from $\#PH$ to $\#P$. *Theoret. Comput. Scie.*, 100:205–221, 1992.

[72] G. T. Toussaint. Solving geometric problems with the rotating calipers. In *Proceedings of IEEE MELECON'83*, 1983.

[73] J. F. Traub, G. W. Wasilkowski, and H. Wozniakowski. *Information-based complexity*. Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[74] A. M. Turing. On computable numbers: With an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 1936.

[75] A. M. Turing. On computable numbers, with an application to the "Entscheidungsproblem". *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.

[76] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[77] K. Weihrauch. *Computability*, volume 9 of *EATCS Monographs on Theoretical Computer Science*. Springer, Berlin, 1987.

[78] K. Weihrauch. *Computable Analysis*. Springer-Verlag, Berlin, 2000.

[79] J. C. Yakoubsohn. Numerical analysis of a bisection-exclusion method to find zeros of univariate analytic functions. *J. Complex.*, 21(5):652–690, 2005.

[80] C. Yap. Theory of real computation according to egc. In *Dagstuhl Seminar on "Reliable Implementation of Real Number Algorithms: Theory and Practice"*, 2006.

[81] C. K. Yap. Robust geometric computation. *Handbook of discrete and computational geometry*, pages 653–668, 1997.

[82] F. Yu. On the representations of $NC$ and Log-space real numbers. *The 13th Annual International Computing and Combinatorics Conference (COCOON 2007), Banff, Canada, July 2007*.

[83] F. Yu. On some complexity issues of nc analytic functions. In J. yi Cai, S. B. Cooper, and A. Li, editors, *TAMC*, volume 3959 of *Lecture Notes in Computer Science*, pages 375–386. Springer, 2006.

[84] F. Yu. On the complexity of the pancake problem. *Mathematical Logic Quarterly*, (to appear).

[85] F. Yu, A. W. Chou, and K.-I. Ko. On the complexity of finding circumscribed rectangles and squares for a two-dimensional domain. *J. of Complexity*, 22(6):803–817, December 2006.

[86] X. Zheng. Binary enumerability of real numbers. In T. Asana, H. Imai, D. Lee, S.-i. Nakano, and T. Tokuyama, editors, *Computing and Combinatorics*, volume 1627 of *Lecture Notes in Computer Science*, pages 300–309, Berlin, 1999. Springer. 5th Annual Conference, COCOON'99, Tokyo, Japan, July 1999.

[87] N. Zhong. Derivatives of computable functions. *Mathematical Logic Quarterly*, 44:304–316, 1998.