LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-481

ON THE COMPLEXITY OF COMPUTING ALGEBRAIC FUNCTIONS

Yishay Mansour

September 1990

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

MIT/LCS/TR-481

ON THE COMPLEXITY OF COMPUTING ALGEBRAIC FUNCTIONS

Yishay Mansour

September 1990

On the Complexity of Computing Algebraic Functions

by

Yishay Mansour

Submitted to the Department of Electrical Engineering and Computer Science
on June, 1990, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This research addresses the problem of proving lower bounds on the complexity of algebraic computations involving the floor operation. The model of computation considered is a computation tree with the set of basic operations $\{+,-,*,/,\lfloor\cdot\rfloor,\geq\}$. The constants available to the computation are 0 and 1, and every other constant needs to be generated explicitly. The problems that are considered may be broken down into the following categories:

1. *Functions of a single input.* A general lower bound technique is developed for a class of functions that have as their input a single n-bit integer. A characterization of the functions to which this technique applies is given. The characterization is general, and applies to many natural functions, such as *perfect square root* (deciding if the square root of the input is integral or not) and computing the value of $\lfloor \log\log x \rfloor$. Every function that with this characterization requires at least $\Omega(\sqrt{\log n})$ operations.

   Upper bounds are given which use the floor function in non-trivial ways. The most surprising result is that the computation of 2^{2^k} can be accelerated in the presence of the floor function. Using only rational operations (i.e. $\{+,-,*,/\}$), it requires $O(k)$ operations to compute (by repeated squaring), and this bound is tight (for rational operations). In contrast, we show that with the use of the floor operation and an additional input greater than 2^{2^k}, the number 2^{2^k} can be generated in $\Theta(\sqrt{k})$ operations. Using the upper bounds developed, an $O(\sqrt{\log n})$ upper bound for computing $\lfloor \log\log x \rfloor$ is given (where x is an n-bit integer). This upper bound matches the lower bound proved for this function.

2. *Functions of two inputs.* A very natural example of a function that has two integer inputs is the *Greatest Common Divisor* function. Grötschel, Lovász and Schrijver posed as an open problem the problem of showing that computing the greatest

Acknowledgments

I would like to thank all the people who made my graduate study so enjoyable. First I would like to thank both of my advisors, Shafi Goldwasser and Baruch Awerbuch, for their encouragement and support during the course of the research.

I am grateful to Shafi Goldwasser for providing the special insight into theoretical computer science, that was an invaluable contribution to this research. The discussions with her have contributed greatly to crystallize my ideas, and provided new ideas and inspiration for future research.

I am grateful to Baruch Awerbuch for the many discussions we had. Discussions that that focused on many issues in communication networks. I would like especially to thank him for sharing with me his ideas even at early stages of the research. His humor always helped to create a relaxed atmosphere.

I would like to express my special thanks to Baruch Schieber and Prasoon Tiwari, with whom I conducted a large portion of the research. The collaboration with both of them was stimulating, exciting and enjoyable.

I would like to thank Ron Rivest who introduced me to the area of machine learning. The course that I took with him during my first term in MIT, and the term paper that I wrote in it, were my first steps in entering the exciting field of theoretical machine learning.

I would like to thank Nancy Lynch for suggesting the problem of communication over a single link, a problem that interested me during much of my graduate years. By working with her I learned the great benefits that could be derived from precise and formal definitions.

It is a great privilege to thank Silvio Micali, who was on my oral exam, area exam and thesis committees. The discussions that we had in his office were always intriguing.

Contents

Chapter 1

Introduction

It is very natural to define the complexity of an algorithm with respect to the number of operations that it performs. Computation trees give a way to formalize such a model. A computation tree, intuitively, has a set of basic operations, each requires unit time, and the complexity of the algorithm is measured with respect to this set of operations. (See Chapter 2 for both motivation and definition of the computation tree model.)

Arithmetic operations, e.g., addition, multiplication, etc, are natural candidates for the set of allowable unit time operations. Deriving lower bounds for computations that involve arithmetic operations has received much attention. As one would expect, at first lower bounds were developed for limited sets of arithmetic operations, and with time, the results were extended to richer sets of operations. Probably the simplest model is one that has only comparisons, which is used to show an $\Omega(n \log n)$ lower bound for sorting (see [Knu81]). A considerably richer model is one that allows linear operations in addition to comparisons. Lower bounds for computations with linear operations can be found in [Rei71, Dob76, DL78, DL80, Yao75, YR80]. As for rational operations, i.e. $\{+, -, *, /\}$, the first lower bound is [Rab72], which is an information theoretic lower bound.

inputs. If the size of numbers is not restricted then "too many" functions are computed with a polynomial number of operations. For example, any PSPACE-computation can be simulated in polynomial time [BMS81, PS76, Sim81]. Moreover, it is known that hard problems, e.g., factoring, can be solved in *linear* time [Sha79]. Restricting the size of the intermediate results to be polynomial in the size of the inputs ensures that strongly polynomial time is contained in polynomial time.

Lower bounds techniques for computations that use the floor operation have been much less successful than for rational operations. Perhaps one reason for the lack of progress in this area is that this set of operations does not possess the "nice" algebraic properties that rational functions possess.

Previously there have been a few lower bound techniques developed to handle the floor operation. In [BJM88], a computation tree that operates on real numbers and has the floor operation in its repertoire was studied. They show, using topological arguments, that there are certain classes of languages which can not be decided by analytic computation trees. [JMW89], compares the expressive power of computation trees with various sets of operations, and proves lower bounds for computation trees with the operations $\{+, -, DIV_c\}$, where DIV_c denotes integer division by constants. Based on proof techniques of [JMW89], [Bas90a] shows an $\Omega(n/\log n)$ lower bound for computing the GCD, using $\{+, -, \lfloor\cdot\rfloor, \times_c, /_c\}$, where $\times_c$ and $/_c$ are multiplication and division by constants. (Note that in such a model the mod operation cannot be performed in $O(1)$ operations.) The work of [IMR83] discusses the relation of the floor operation and indirect addressing. In [DO85, Jia79], an $\Omega(n \log n)$ lower bound for sorting rational numbers, using a restricted floor operation, is given.

The aim of our research is to establish lower bounds for computations that use the floor operation, in addition to rational operations. The main motivation for considering the floor operation, is that it is simple to implement in practice, and that it exists as a basic operation in most computers and programming languages.

the average error is less than ϵ, requires at least $\Omega(\sqrt{\log\log\frac{1}{\epsilon}})$ operations. (In the lower bound we assume that the computation receives only the input $x \in [1,2]$, as in the case of Newton iterations, and tries to produce the best approximation it can, given a bound on the number of operations.) In a somewhat different setting, we show how to approximate the s^{th} root, for any $x \in [1,2]$. The approximated value, for any input $x \in [1,2]$, is at most ϵ away from $\sqrt[s]{x}$, and the number of operations is $O(\sqrt{\log\log\frac{1}{\epsilon}})$. (In the upper bound, the computation receives both the input $x \in [1,2]$ and the accuracy parameter ϵ.)

All our lower bounds assume that we have to generate all the constants that are used in the computation explicitly. We show that without this restriction any polynomial could be computed in $O(1)$ operations (independent of its degree). In order to avoid any possible confusion, we quantify our lower bounds by stating that the computation has initially only the constants $\{0,1\}$.

The research in this work was done in collaboration with Baruch Schieber and Prasoon Tiwari, and preliminary versions of it appear in [MST88, MST89b, MST89a].

This thesis is organized as follows. In Chapter 2 we define the computation model and prove a few results concerning polynomials and rational functions. In Chapter 3 we develop the proof technique for one variable functions. In Chapter 4 we extend the technique from one variable to two variables, and show the lower bound for computing the greatest common divisor of two integers. In Chapter 5 we show the lower bounds for approximation. In Chapter 6 we show the upper bounds. In Chapter 7 we summarize the results and suggest directions for future research.

Chapter 2

Preliminaries

2.1 Computation Tree Model

In this section, we define the computation tree model. Let us motivate the definition chosen. Consider a specific computer, it has some set of instructions that are built into it. A program that runs on that computer uses this set of operations, usually referred to as "machine instructions". A natural measure of the time complexity of a program is the number of machine instructions that it performs. Although this approach assumes that each machine instruction requires the same amount of time, in many cases this gives a good approximation.

This means that it is natural to fix some set of operations, and consider them as *basic operations.* The time complexity of computing a certain function is the number of basic operations performed.

We would like a formal model that would make it "easy" to define time complexity. If the set of basic operations does not include a branching operation, then the number of operations performed, on any input, is simply the number of operations in the program.

Definition 1 *A* computation tree *T consists of a labeled binary tree, and a set of operations OP. Each vertex of the tree is labeled in one of the following ways.*

- Input vertices: *An input vertex is labeled by a certain input, e.g. x_1. Each input vertex has at most one child.*
- Computation vertices: *Each computation vertex u is labeled with an operation $f_u = \circ(v_1, \ldots, v_k)$, where $\circ \in OP$, and v_i is either a computation or an input vertex that appears on the path from the root to node u. Each computation vertex has at most one child.*
- Comparison vertices: *Each comparison vertex u is labeled with $v_1 \leq v_2$, where, again, v_i is either a computation or an input vertex that appears on the path from the root to node u. Each comparison vertex has two children.*

Given an input the computation tree defines a *computation.* The computation traverses a path from the root to a leaf and assigns values to the input and computation vertices on the path in the following way.

The computation starts at the root of the tree T, and proceeds as follows:

1. When the computation arrives at an input vertex u, labeled by x_i, it assigns u the value of the input x_i, and continues to the child of u.
2. When the computation arrives at a computation vertex u, labeled by $f_u = op(v_1, \ldots, v_k)$, it assigns u the value $val(u)$, which is $op(val(v_1), \ldots, val(v_k))$, where $val(v_i)$ is the value of vertex v_i. Note, that since v_i appears on the path from the root to u, the computation already assigned some value to v_i. The computation continues to the child of u.
3. When the computation arrives at a comparison vertex u, labeled with $v_1 \leq v_2$, it proceeds to the left child of u, if $val(v_1) \leq val(v_2)$, and to the right child of u, otherwise. (A comparison node does not have a value.)

2.2 Polynomials and rational expressions

This section includes some basic definitions and results about polynomial and rational expressions. These results will be used extensively in the following chapters.

The *degree* of a polynomial $P(x_1, \ldots, x_k)$ with respect to a variable x_i, denoted $deg_{x_i}(P)$, is the maximum exponent of x_i appearing in any monomial of $P(x_1, \ldots, x_k)$. The *degree* of P, denoted $deg(P)$, is $\max_i deg_{x_i}(P)$.

The *size* of the coefficients plays a crucial role in the lower bounds that we prove later. For this reason we add another parameter, which is the *maximum coefficient* of P, denoted $max\text{-}coef(P)$. The value of $max\text{-}coef(P)$ is defined to be the maximum among the absolute values of the coefficients of P.

We extend the notion of degree and maximum coefficient of a polynomial to a set of polynomials, in a rather straightforward way. For a set Λ of polynomials, the *degree* of Λ, denoted $deg(\Lambda)$, is $\max_{P \in \Lambda} deg(P)$. Similarly, the *maximum coefficient* of Λ, denoted $max\text{-}coef(\Lambda)$, is $\max_{P \in \Lambda} max\text{-}coef(P)$.

The following lemma relates the degree and maximum coefficient of polynomials and the operations $\{+, -, *\}$.

Lemma 2.1 *Let $P(x_1, \ldots, x_k)$ and $Q(x_1, \ldots, x_k)$ be two multivariate polynomials. Then,*

1. $deg(P \pm Q) \leq \max\{deg(P), deg(Q)\}$,

2. $max\text{-}coef(P \pm Q) \leq max\text{-}coef(P) + max\text{-}coef(Q)$,

3. $deg(P * Q) \leq deg(P) + deg(Q)$, *and*

4. $max\text{-}coef(P * Q) \leq (1 + \min\{deg(P), deg(Q)\})^k max\text{-}coef(P) max\text{-}coef(Q)$.

A lexicographic order on polynomials

We define a lexicographic order on the set of multivariate polynomials. For this purpose, we use the following lexicographic order on the set of multivariate monomials.

Definition 2 *For two monomials $cx_1^{i_1} \dots x_k^{i_k}$ and $dx_1^{j_1} \dots x_k^{j_k}$, the relation $cx_1^{i_1} \dots x_k^{i_k} \succ dx_1^{j_1} \dots x_k^{j_k}$ holds if either (1) there exists an m, such that $i_m > j_m$, and for any $l < m$, $i_l = j_l$, or (2) $i_l = j_l$, for $1 \leq l \leq k$, and $|c| > |d|$.*

We say that a polynomial is written in its *normal form* if it is written as a minimal sum of monomials, and these monomials are sorted in descending lexicographic order. We assume that all polynomials are written in their normal form.

Definition 3 *For two multivariate polynomials $P(x_1, \dots, x_k)$ and $Q(x_1, \dots, x_k)$, $P(x_1, \dots, x_k) \succ Q(x_1, \dots, x_k)$ if, when written in their normal forms, there exists some $i \geq 1$, such that* (the i-th monomial in P) $\succ$ (the i-th monomial in Q)*, and all the monomials preceding it are identical in both P and Q.*

Given a polynomial $P(x_1, \dots, x_k)$, let the *leading monomial* of P be the first monomial in the normal form of $P(x_1, \dots, x_k)$. Let the *leading coefficient* of $P(x_1, \dots, x_k)$ be the coefficient of this monomial. Define

$$sign(\alpha) = \begin{cases} +1 & \text{for} \quad \alpha > 0 \\ 0 & \text{for} \quad \alpha = 0 \\ -1 & \text{for} \quad \alpha < 0 \end{cases}$$

The *sign* of a polynomial P, denoted by *sign*(P), is the sign of the leading coefficient of P. Note that *sign*(P) $= 0$ if and only if P is the zero polynomial. A *sign* of a rational expression $R(x) = \frac{P(x)}{Q(x)}$, is *sign*(P) *sign*(Q).

To complete the proof we show that if $b > \pi_2(P) = \frac{M+L}{L}$ then $(1 - \sum_{k=1}^{m} \frac{|L_k|}{Lb^k}) > 0$, or equivalently, $\sum_{k=1}^{m} \frac{|L_k|}{Lb^k} < 1$. This later inequality follows from

$$\sum_{k=1}^{m} \frac{|L_k|}{Lb^k} \le \frac{M}{L} \sum_{k=1}^{m} b^{-k} < \frac{M}{L} \sum_{k=1}^{m} \left(\frac{L}{M+L}\right)^k < \frac{M}{L} \frac{L}{M+L} \frac{1}{1-\frac{L}{M+L}} = 1.$$

Similarly, we can prove the Lemma for the case $L < 0$. □

From Lemma 2.3 we can infer a sufficient condition for univariate polynomials.

Corollary 2.4 *For each polynomial $P(y)$, there exist a positive integer $\pi(P)$ such that for all $b > \pi(P)$, $sign(P(b))$ equals $sign(P)$. Furthermore, $\pi(P) \le \frac{max\text{-}coef(P)}{|L|} + 1$, where L is the leading coefficient of P.*

Corollary 2.4 enables to argue along the following lines. Consider a comparison between $Q(x)$ and $P(x)$, where both Q and P are polynomials. By Corollary 2.4, there exists a value $e = \pi(P-Q)$, such that for $x > e$, either $P(x)-Q(x) = 0$, $P(x)-Q(x) > 0$, or $P(x) - Q(x) < 0$. This means that, for $x > e$, the value of the comparison between Q and P is fixed. In a similar way, Lemma 2.3 allows us to argue about the comparison of bivariate polynomials.

Polynomial division

The lemma below shows how the degree and the maximum coefficients of bivariate polynomials are affected, when we perform polynomial division. Note that the following lemma is restricted to the case that the leading monomial has only one variable.

Lemma 2.5 *Let $P(x,y)$ and $Q(x,y)$ be two bivariate polynomials with integer coefficients. Let $max\text{-}coef(P) = M$, $max\text{-}coef(Q) = N$, and $\delta = \max\{-1, deg_x(P) - deg_x(Q)\}$, If Lx^d is the leading monomial of $Q(x,y)$, where $d \ge 1$, then there exists $A(x,y)$ and*

Therefore we can bound the maximum coefficient of R, using the inductive hypothesis, as follows,

$$\begin{aligned} \textit{max-coef}(R) &\leq (2+\textit{deg}_y(Q))^{\delta}\textit{max-coef}(S)N^{\delta} \\ &\leq (2+\textit{deg}_y(Q))^{\delta}\left((2+\textit{deg}_y(Q))MN\right)N^{\delta} \\ &\leq (2+\textit{deg}_y(Q))^{\delta+1}MN^{\delta+1}. \end{aligned}$$

Furthermore, $\textit{deg}_x(R) < \textit{deg}_x(Q)\{= d\}$, and $\textit{deg}_y(R) \leq \textit{deg}_y(S) + (\delta - 1)\textit{deg}_y(Q) \leq \textit{deg}_y(P) + \delta'\textit{deg}_y(Q)$.

In a similar way the bounds on *max-coef*(A) and *deg*(A) follow. □

For most of our applications the following simplified version of Lemma 2.5 would be sufficient.

Corollary 2.6 *Let $P(x,y)$ and $Q(x,y)$ be two bivariate polynomials such that deg(Q), deg$(P) \leq D$, and max-coef(P), max-coef$(Q) \leq M$. If Lx^d is the leading monomial of of $Q(x,y)$, then there exists $A(x,y)$ and $R(x,y)$ such that*

$$P(x,y) = \frac{1}{L^{\delta+1}}A(x,y)Q(x,y) + \frac{1}{L^{\delta+1}}R(x,y),$$

where $\delta = \max\{-1, \textit{deg}_x(P) - \textit{deg}_x(Q)\} \leq D$. Furthermore, $A(x,y)$ and $R(x,y)$ are polynomials with integer coefficients, that satisfy,

1. *max-coef$(R) \leq ((D+2)M)^{D+2}$,*
2. *max-coef$(A) \leq ((D+2)M)^{D+1}$,*
3. *$\textit{deg}_x(R) < D$, and $\textit{deg}_y(R) \leq (D+1)D$.*
4. *$\textit{deg}_x(A) \leq D$, and $\textit{deg}_y(A) \leq D$.*

Chapter 3

One Variable Functions

3.1 Overview

Proving that a polynomial cannot compute a certain boolean function, using only a certain number of $\{+,-,*\}$ operations, can be done by arguing about the degree of the polynomial. In order to make the case more interesting, assume that we relax the requirement on the output of the polynomial such that it evaluates to a non-negative number if and only if the computed function is TRUE on that input. On the one hand, the number of alternations between TRUE and FALSE of the output of the computed function gives a lower bound on the degree of the polynomial. On the other hand, in order to evaluate a polynomial of degree d, one must perform at least $\log d$ multiplications. This degree argument is independent of the size of the coefficients of the polynomial, and applies even if the input is restricted to be integer. The degree argument can be extended also to rational functions, and to computation tree with rational operations (i.e. $\{+,-,*,/\}$). Stockmeyer [Sto76] showed, using a similar argument, an $\Omega(n)$ lower bound for the depth of any computation tree that computes the parity, for any n bit integer, using rational operations.

Doing it in a straightforward way may cause a "huge" increase in the degree.

For example, consider the expression $\left\lfloor \frac{x}{2} \right\rfloor$, where x is an n bit integer. If R is a rational function such that $R(x) = \left\lfloor \frac{x}{2} \right\rfloor$, then R has degree 2^{n-1}. We are interested in a transformation that keeps the degree and the coefficients "small". In this specific case, a good solution would be to set $\left\lfloor \frac{x}{2} \right\rfloor = 0$, and restrict x to integers that are divisible by 2, i.e. only even integers.

The general solution has a similar flavor. We restrict the input, so we can set the value of the floor operation equal to a rational function. In this transformation, the degree remains the same, and the size of the coefficients may grow at most exponentially in the degree. Based on this technique, we show how to derive lower bounds for a decision tree.

In Section 3.3 we show how to derive lower bound for various functions, using the general technique. Although our proof technique is for decision problem we can use it to derive lower bounds for computation problems as well. In the cases that we consider, the decision problem is the parity of the output of the computation problem. Since the floor is a basic operation, we can compute the parity from the output in $O(1)$. This implies that for such decision problems, the computation problem is at least as hard as the decision problem.

3.2 Proof Technique

We start by defining the subsets of the input that will be used in our proof.

Definition 4 *Let the set $S(n, \lambda)$ be the set of all n-bit integers that are multiples of an integer λ.*

The interesting case of the proof technique is the floor operation. This is by far more involved than the previous cases. After proving this lemma, we show how to combine the techniques we developed to derive a lower bound for decision trees. Recall that we restrict the floor operation to non-negative inputs.

Lemma 3.4 *Let P be a rational expressions with integer coefficients, of degree at most D and maximum coefficient at most M, defined over $S(n, \lambda)$, such that $P(x) \geq 0$ for any $x \in S(n, \lambda)$. There there exists $\pi \leq (2M)^{D+2}$, and a rational expression Q with integer coefficients, where deg(Q) $\leq D$ and max-coef(Q) $\leq (2M)^{D+1}$, such that for any $x \in S(n, \lambda\pi)$, $Q(x) = \lfloor P(x) \rfloor$.*

Proof: If $sign(P) = 0$, i.e. P is the zero function, then let $Q(x) = 0/1$ and $\pi = 1$, and the lemma follows. Otherwise, let $P(x) = \frac{P_1(x)}{P_2(x)}$. Without loss of generality we may assume that $sign(P_2) = +1$. We would like to restrict our attention to a subset of the inputs, for which the sign of $P(x)$ is fixed. By Corollary 2.4 for $x \geq \max\{\pi(P_1), \pi(P_2)\}$, $sign(P(x))$ equals $sign(P)$, which equals $sign(P_1)sign(P_2)$. Since the operand of the floor is non-negative, $sign(P) = +1$, which implies that $sign(P_1) = +1$.

We consider three cases of relations between P_1 and P_2.

Case 1: $P_1(x) = P_2(x)$. This is the trivial case. Let $Q(x) = 1/1$, and the lemma follows.

Case 2: $P_1(x) \prec P_2(x)$. Let $B(x) = P_2(x) - P_1(x)$. Since the leading coefficient of P_2 is positive then $sign(B) = +1$. Corollary 2.4 guarantees the existence of a positive integer $\pi(B)$ such that $B(a) > 0$, for all $a > \max\{\pi(P_1), \pi(P_2), \pi(B)\} = \pi$. Since $P_2(x) > 0$, the expression $P_2(x) - P_1(x) > 0$ is equivalent to $1 > \frac{P_1(x)}{P_2(x)} = P(x)$. Since $P(x) \geq 0$, for $x \in S(n, \lambda)$, $1 > \frac{P_1(a)}{P_2(a)} \geq 0$, for $a \in S(n, \lambda\pi)$. We conclude that for any integer $a \in S(n, \lambda\pi)$, $\lfloor P(a) \rfloor = 0$. Let $Q(x) = 0/1$. Since $\pi \leq 2(D+1)M^2 + 1$, the lemma follows.

We need to bound the parameters π, *deg*(Q) and *max-coef*(Q). To bound *deg*(Q) and *max-coef*(Q) it is sufficient to consider the polynomial $\hat{A}(x) + cL^d$. Clearly, *deg*$(\hat{A}(x) + cL^d) \leq D$. The value of *max-coef*(Q) is the maximum between L^d and *max-coef*$(\hat{A}(x) + cL^d)$. By definition, *max-coef*$(\hat{A}(x) + cL^d) \leq$ *max-coef*$(A) + L^d$. Using Corollary 2.7, *max-coef*$(A) < 2^D M^{D+1}$. Therefore *max-coef*$(Q) \leq (2M)^{D+1}$.

To bound the value of π, we bound $\pi(B)$; the bounds for $\pi(\gamma P_2 + R)$, $\pi(P_1)$, and $\pi(P_2)$ are smaller, and derived similarly. Recall that $L^d B(x) = L^d P_2(x) - (\gamma P_2(x) + R(x))$. Therefore, *max-coef*$(L^d B) \leq (L^d - \gamma)$ *max-coef*(P_2) + *max-coef*(R) $< M^{D+1}M + 2^{D+1}M^{D+2}$. Since $L > 0$, for any x, *sign*$(B(x))$ = *sign*$(L^d B(x))$. Hence, in order to bound $\pi(B)$ it is sufficient to bound $\pi(L^d B)$. Since $L^d B(x)$ is a polynomial with integer coefficients, its leading coefficient is at least one. Therefore, by Corollary 2.4, $\pi(L^d B) \leq$ *max-coef*$(L^d B)$ +1. By requiring that π is divisible by L^d, we may add $L^d - 1$ to $\pi(B)$, therefore, $\pi \leq \pi(L^d B) + 1 + (L^d - 1) \leq (2M)^{D+2}$.

Subcase 2: $\gamma = 0$. Clearly $R(x) \prec L^d P_2(x)$. Using Corollary 2.4, let π be the minimum multiple of L^d such that $\pi \geq \max\ \{\pi(R), \pi(P_1), \pi(P_2), \pi(L^d P_2 \pm R)\}$. Then, for all $a > \pi$, $\frac{R(a)}{L^d P_2(a)} = 0$, if *sign*$(R) = 0$, $0 < \frac{R(a)}{L^d P_2(a)} < 1$, if *sign*$(R) = +1$, and $-1 < \frac{R(a)}{L^d P_2(a)} < 0$, if *sign*$(R) = -1$. Recall that the free term of A is $L^d c + \gamma$. Let $\tilde{c}$ be c if *sign*$(R) \geq 0$, and $c - 1$ if *sign*$(R) = -1$. We conclude that for each $a \in S(n, \lambda\pi)$, $\left\lfloor \frac{P_1(a)}{P_2(a)} \right\rfloor = L^{-d}\hat{A}(a) + \tilde{c}$. Let

$$Q(x) = \frac{\hat{A}(x) + \tilde{c}L^d}{L^d} = \lfloor P(x) \rfloor \text{ for } x \in S(n, \lambda\pi).$$

We can bound the parameters π, *deg*(Q) and *max-coef*(Q) as in the previous subcase.

Since one of the above relations has to hold between P_1 and P_2, the lemma follows. □

To summarize, so far we considered each operation by itself. Lemma 3.4 guarantees that each floor operation can be replaced by an evaluation of a rational expression.

3. Let $\Sigma_i = \{F_{v_j}, G_{v_j} \mid v_j$ is a computation vertex, $j \leq i\}$. Define $D_i = deg(\Sigma_i) + 1$ and $M_i = max\text{-}coef(\Sigma_i)$. Then, $D_i < 2^i$, and $\max\{\lambda^{(i)}, M_i\} < 2^{2^{4i^2}}$.

We show how to define $\lambda^{(i+1)}$ and how to choose the vertex v_{i+2} such that Properties 1-3 will be satisfied by the set $S(n, \lambda^{(i+1)})$ and the prefix of $\mathcal{P}$ that starts at v_1, and ends at v_{i+2}. The parameter $\lambda^{(i+1)}$ will be a multiple of $\lambda^{(i)}$. By Claim 3.1 this implies that $S(n, \lambda^{(i+1)}) \subseteq S(n, \lambda^{(i)})$, therefore, by the induction hypothesis we have that Properties 1-3 are satisfied by the set $S(n, \lambda^{(i+1)})$ and the prefix of $\mathcal{P}$ that starts at v_1 and ends at v_{i+1}. In order to complete the proof of the lemma we need to show that (a) there exists an outgoing edge of v_{i+1} such that for each input $a \in S(n, \lambda^{(i+1)})$ the computation follows this edge, and (b) Properties 2-3 are satisfied also for the vertex v_{i+1} and the set $S(n, \lambda^{(i+1)})$.

By the definition of the tree T, the vertex v_{i+1} is either an input vertex, a comparison vertex or a computation vertex. For an input vertex both (a) and (b) hold trivially.

For a comparison vertex we need to show that (a) holds. Let the comparison be $g \geq h$, such that $g, h \in \{0,1\} \cup \{f_{v_j} \mid v_j$ is a computation vertex, $j \leq i\}$. By the induction hypothesis both h and g can be represented as rational expressions of degree at most $D_i \leq 2^i$, and maximum coefficient $M_i \leq 2^{2^{4i^2}}$. By Lemma 3.2, there is a $\lambda^{(i+1)} \leq 2\lambda^{(i)}(D_i + 1)M_i^2 + \lambda^{(i)} \leq 2^{2^{4(i+1)^2}}$. For a comparison vertex (b) holds trivially.

For a computation vertex (a) holds trivially, so we need to show only that (b) holds. At a computation vertex, either $f_{v_{i+1}} = g \circ h$, for $\circ \in \{+, -, *, /\}$, or $f_{v_{i+1}} = \lfloor g \rfloor$ is evaluated, where, $g, h \in \{0,1\} \cup \{f_{v_j} \mid v_j$ is a computation vertex, $j \leq i\}$. If $\circ \in \{+, -, *, /\}$, then the induction step follows from Lemma 3.3. If $f_{v_{i+1}} = \lfloor g \rfloor$, then, by Lemma 3.4, there is a rational expression Q, such that $Q = \lfloor g \rfloor$. By Lemma 3.4, $D_{i+1} \leq D_i \leq 2^i$, $M_{i+1} \leq (2M_i)^{D_i+1} \leq 2^{(1+2^{4i^2})(2^i+1)} \leq 2^{2^{4(i+1)^2}}$, and $\lambda^{(i+1)} \leq \lambda^{(i)}(2M_i)^{D_i+2} \leq 2^{2^{4(i+1)^2}}$. □

The following corollary states that for $M(n) = \Omega(2^{n^\epsilon})$, the lower bound is still $\Omega(\sqrt{\log n})$. This corollary is used later to derive an $\Omega(\sqrt{\log n})$ lower bound for various functions.

Corollary 3.7 *Let f be an $M(n)$-invariant function, such that $M(n) = \Omega(2^{n^\epsilon})$ for some fixed $\epsilon > 0$. Then any computation tree with $OP = \{+, -, *, /, \lfloor\cdot\rfloor\}$, that computes $f(x)$, for all n-bit integers, must have depth $\Omega(\sqrt{\log n})$.*

Using Corollary 3.7 we show lower bounds for the following problems.

1. Decide if $\lfloor \log a \rfloor$ is odd or even, for any n-bit integer a. (choose $M(n) = 2^{n-1}$, $a_1 = \lambda$ and $a_2 = 2\lambda$.)
2. Decide if $\lfloor \log \log a \rfloor$ is odd or even, for any n-bit integer a. (choose $M(n) = 2^{n/2}$, $a_1 = \lambda$ and $a_2 = \lambda^2$.)
3. Decide if $\sqrt{a}$ is an integer. (choose $M(n) = 2^{n/2-1}$, $a_1 = \lambda^2$ and $a_2 = 2\lambda^2$.)

Theorem 3.6 gives an $\Omega(\sqrt{\log n})$ lower bound on the depth of any decision tree with $OP = \{+, -, *, /, \lfloor\cdot\rfloor\}$, that solves the above problems.

At first sight it seems that all of the above lower bounds are for decision problems, and may be unrelated to the similar computation problems. In general this may be the case, but our examples have the property that the decision problem is closely related to the corresponding computational problem. For all the above decision problems, given a solution to the corresponding computation problem, one can decide the decision problem in $O(1)$ operations.

Chapter 4

Lower bound for GCD

4.1 Overview

In this chapter we extend the proof technique developed in Chapter 3 to functions with two variables and apply it to show a lower bound for computing the greatest common divisor of two integers. Although this chapter does not depend on chapter 3, it is advisable that the reader would read chapter 3 prior to reading this chapter.

Before explaining the lower bound technique for functions of two variables, we argue why the technique for one variable does not extend immediately to two variables. The most obvious reduction would be to concatenate the two inputs, each of size n bits, to one input of size $2n$ bits, and define the function with respect to this single variable. In order for the reduction to work, we need to argue that given this input, we first extract the two n bit integers, and continue the computation using them. The problem is that in order to extract the two integers one may be required to perform $\Omega(\log n)$ operations. This $\Omega(\log n)$ additive factor would be more than the lower bound that we can prove for the one variable function.

4.2 Inputs structure

We start by defining the subsets of inputs that we will consider in the proof. Recall that for the one variable case, the subsets of the input that we considered were all the integers divisible by some integer. Unfortunately, the structure of the subsets of the inputs is substantially more complex in the two variable case.

Definition 6 *Let r, α_2 and α_3 be non-negative integers, and α_1 a positive integer. Let $\Delta = (\delta_1, \delta_2, \ldots, \delta_r)$ and $\Lambda = (\lambda_1, \lambda_2, \ldots, \lambda_r)$ be r-dimensional vectors of positive integers. For positive integers u_0, u_1, u_r, and u_{r+1}, the pair (u_0, u_1) is $< r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda >$-generated by the pair (u_r, u_{r+1}) if there exist positive integers $u_2, u_3, \ldots, u_{r-1}$ such that:*

$$\begin{aligned}
u_0 &= \lambda_1(u_1)^{\delta_1} + u_2, && (4.1)\\
u_1 &= \lambda_2(u_2)^{\delta_2} + u_3, \\
&\vdots \\
u_i &= \lambda_{i+1}(u_{i+1})^{\delta_{i+1}} + u_{i+2}, \\
&\vdots \\
u_{r-1} &= \lambda_r(u_r)^{\delta_r} + u_{r+1}, \\
u_r &> (u_{r+1})^{\alpha_1}, && (4.2)\\
u_{r+1} &> \alpha_2, \\
u_r &\equiv u_{r+1} \equiv 1 \pmod{\alpha_3}. && (4.3)
\end{aligned}$$

In this case, (u_r, u_{r+1}) is the $< r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda >$-generator of (u_0, u_1), and $u_2, u_3, \ldots, u_{r+1}$ is the $<r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda>$-generating sequence for (u_0, u_1).

The following lemma shows that when we add another variable, and thus adding another element to Δ and Λ, we restrict the "allowable" inputs to be a subset of the previous "allowable" inputs.

Lemma 4.2 (The Containment Property): *Let Δ' and Λ' be $(r+1)$-dimensional vectors of positive integers, obtained from r-dimensional vectors Δ and Λ by appending positive integers δ and λ, respectively. Then, $S(r+1, \alpha_1, \alpha_2, \alpha_3, \Delta', \Lambda') \subseteq S(r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda)$, provided $\delta \geq \alpha_1$ and $\lambda \equiv 0 \pmod{\alpha_3}$.*

Proof: Suppose that $(a_0, a_1) \in S(r+1, \alpha_1, \alpha_2, \alpha_3, \Delta', \Lambda')$, and let $a_2 > a_3 > \ldots > a_r > a_{r+1} > a_{r+2}$ be its generating sequence. By definition, $a_r = \lambda a_{r+1}^{\delta} + a_{r+2}$. Therefore, $a_r \equiv 1 \pmod{\alpha_3}$, and $a_r > a_{r+1}^{\alpha_1}$. In addition, $a_{r+1} > a_{r+2}^{\alpha_1} > \alpha_2$. Hence, $(a_0, a_1) \in S(r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda)$. □

4.3 The proof technique

As in the proof technique for one variable, we start by showing how to handle a comparison. We show that we can restrict the inputs to the comparison, so as to ensure that the value of the comparison is fixed.

Lemma 4.3 *Let $P(x,y)$ and $Q(x,y)$ be two rational expressions with integer coefficients, of degree at most D and maximum coefficient at most M, defined over $S(0, \alpha_1, \alpha_2, \alpha_3)$. There exists $\alpha_1 \leq \alpha_1' \leq \max\{\alpha_1, 2D+1\}$, and $\alpha_2 \leq \alpha_2' \leq \max\{\alpha_2, 2(D+1)^2M^2+1\}$, such that for all $(a,b) \in S(0, \alpha_1', \alpha_2', \alpha_3)$, the comparison between $P(a,b)$ and $Q(a,b)$ is determined (i.e. either $P(a,b) = Q(a,b)$, $P(a,b) > Q(a,b)$, or $P(a,b) < Q(a,b)$, for $(a,b) \in S(0, \alpha_1', \alpha_2', \alpha_3)$), and $S(0, \alpha_1', \alpha_2', \alpha_3) \subset S(0, \alpha_1, \alpha_2, \alpha_3)$.*

Proof: If $P(x,y) = Q(x,y)$, for $(x,y) \in S(0, \alpha_1, \alpha_2, \alpha_3)$, then let $\alpha_1' = \alpha_1$ and $\alpha_2' = \alpha_2$, and the lemma follows.

1. *There is a rational expression $Q(x,y)$ with integer coefficients, and integers α_1', α_2', π such that*

   $$\lfloor P(x,y) \rfloor = Q(x,y)$$

   for each $(x,y) \in S(0, \alpha_1', \alpha_2', \pi\alpha_3)$. Furthermore, $\max\{\alpha_1',\ \mathit{deg}(Q),\ \} \leq 4D^4$*, and* $\max\{\pi\alpha_3, \alpha_2', \mathit{max\text{-}coef}(Q), \} \leq 2(4D^2M)^{2D^3}$*, or*

2. *There is a rational expression $Q(y,z)$ with integer coefficients, and integers α_1', α_2', π, λ and δ such that*

   $$\lfloor P(x,y) \rfloor = Q(y,z)$$

   where (y,z) is the generator of $(x,y) \in S(1, \alpha_1', \alpha_2', \pi\alpha_3, \{\delta\}, \{\lambda\})$. Furthermore, $\max\{\alpha_1',\ \mathit{deg}(Q),\ \delta\} \leq 4D^4$*, and* $\max\{\pi\alpha_3, \alpha_2', \mathit{max\text{-}coef}(Q), \lambda\} \leq 2(4D^2M)^{2D^3}$.

Proof: Let $d_x = \mathit{deg}_x(P_2)$, $d_y = \mathit{deg}_y(P_2)$, and let $Lx^{d_x}y^l$ be the leading monomial of $P_2(x,y)$. Without loss of generality we may assume that $L > 0$, i.e. $\mathit{sign}(P_2) = +1$.

In order to simplify the proof, we would like to fix the inputs such that $\mathit{sign}(P(x,y)) = \mathit{sign}(P)$. By Lemma 2.3, there are two integers $\pi_1(P_1)$ and $\pi_2(P_1)$, such that for each $(u,v) \in S(0, \pi_1(P_1), \pi_2(P_1), 1)$, $\mathit{sign}(P_1(u,v)) = \mathit{sign}(P_1)$. By a similar argument there are $\pi_1(P_2)$ and $\pi_2(P_2)$ such that for each $(u,v) \in S(0,\ \pi_1(P_2),\ \pi_2(P_2),\ 1)$, $\mathit{sign}(P_2(u,v)) = \mathit{sign}(P_2) = +1$. Since the input to the floor operation is non-negative, $\mathit{sign}(P) = +1$. Recall that $\mathit{sign}(P) = \mathit{sign}(P_1)\mathit{sign}(P_2)$; this implies that $\mathit{sign}(P_1) = +1$. Let $\hat{\alpha}_1 = \max\{\alpha_1, \pi_1(P_1), \pi_1(P_2)\}$ and $\hat{\alpha}_2 = \max\{\alpha_2, \pi_2(P_1), \pi_2(P_2)\}$. Note that for each $(u,v) \in S(0, \hat{\alpha}_1, \hat{\alpha}_2, 1)$, both $P_1(u,v) > 0$ and $P_2(u,v) > 0$. Since the coefficients of P_1 and P_2 are integers, by Lemma 2.3, $\hat{\alpha_1} \leq D + 1$ and $\hat{\alpha_2} \leq M + 1$.

In the remainder of the proof we consider the following five cases in the following order:

1. $P_1(x,y) = P_2(x,y)$.
2. $P_1(x,y) \prec P_2(x,y)$.

and γ is a non-negative integer such that $0 \leq \gamma < L$. Let $\pi = L$, $\alpha_1' = \hat{\alpha}_1$, and $\alpha_2' = \hat{\alpha}_2$. Then, for each $(u, v) \in S(0, \alpha_1', \alpha_2', \pi\alpha_3)$,

$$P_1(u, v) \bmod L \equiv P_1(1, 1) \bmod L \equiv \gamma,$$

Since $u \equiv v \equiv 1 \bmod L$. By writing the mod operation explicitly, we have,

$$P_1(u, v) = \left\lfloor \frac{P_1(u, v)}{L} \right\rfloor L + \gamma$$

Therefore, for $(x, y) \in S(0, \alpha_1', \alpha_2', \pi\alpha_3)$,

$$\left\lfloor \frac{P_1(u, v)}{L} \right\rfloor = \frac{P_1(u, v) - \gamma}{L} = Q(x, y)$$

Clearly, *deg*$(Q) \leq D$, *max-coef*$(Q) \leq 2M$, and $\pi = L \leq M$.

Case 4: The leading monomial of $P_2(x, y)$ is Lx^{d_x}, i.e., no power of y appears in the leading monomial of $P_2(x, y)$. We use Corollary 2.6 to divide $P_1(x, y)$ by $P_2(x, y)$ as polynomials in x. By Corollary 2.6 there are polynomials $A(x, y)$ and $R(x, y)$ with integer coefficients such that

$$P_1(x, y) = \frac{A(x, y)P_2(x, y) + R(x, y)}{L^d}.$$

Note that since Case 3 does not hold, $deg_x(P_2) \geq 1$. Furthermore, the following properties hold: (i) $d \leq D$, (ii) $deg_x(R) < deg_x(P_2)$. (iii) *max-coef*$(R) \leq ((D + 2)M)^{D+2}$, (iv) *max-coef*$(A) \leq ((D + 2)M)^{D+1}$, (v) $deg_x(R) < D$, and $deg_y(R) \leq (D + 1)D$. (vi) $deg_x(A) \leq D$, and $deg_y(A) \leq D$.

Consider input $(u,v) \in S(0,\alpha_1',\alpha_2',\pi\alpha_3)$. Then, both $V(u,v) < P_2(u,v)$ and $0 < V(u,v)$. Since $P_2(u,v) > 0$, this implies that,

$$0 < \frac{V(u,v)}{P_2(u,v)} = \frac{\gamma P_2(u,v) + R(u,v)}{L^d P_2(u,v)} < 1.$$

We conclude that for each $(x,y) \in S(0,\alpha_1',\alpha_2',\pi\alpha_3)$,

$$\lfloor P(x,y) \rfloor = \frac{A(x,y) - \gamma}{L^d} = Q(x,y).$$

Clearly, $deg(Q) = deg(A) \leq D$, and $max\text{-}coef(Q) \leq max\text{-}coef(A) + M^d \leq ((2+D)M)^{D+1} + M^D \leq 2((D+2)M)^{D+2}$.

Subcase 4.2: $\gamma = 0$. In this subcase we restrict the input, such that the value of $\left\lfloor \frac{\gamma P_2(u,v)+R(u,v)}{L^d P_2(u,v)} \right\rfloor$ is either 0 or -1, depending on $sign(R)$. Define $\tilde{R}(x,y)$ as follows:

$$\tilde{R}(x,y) = \begin{cases} R(x,y) & \text{if } sign(R) \geq 0 \\ L^d P_2(x,y) + R(x,y) & \text{if } sign(R) = -1 \end{cases}$$

Note that,

$$\left\lfloor \frac{\tilde{R}(x,y)}{L^d P_2(x,y)} \right\rfloor + \rho(R) = \left\lfloor \frac{R(x,y)}{L^d P_2(x,y)} \right\rfloor$$

where $\rho(R) = 0$, if $sign(R) \geq 0$, and $\rho(R) = -1$, if $sign(R) = -1$. The leading coefficients of the polynomials $\tilde{R}(x,y)$ and $L^d P_2(x,y) - \tilde{R}(x,y)$ are positive, assuming that $sign(R) \neq 0$ (the case $sign(R) = 0$ is trivial). Recall that $\pi = L^d$. Using Lemma 2.3, let

$$\begin{aligned} \alpha_1' = \max\{\hat{\alpha}_1, \pi_1(\tilde{R}), \pi_1(L^d P_2 - \tilde{R})\} &\leq D(D+1) \\ &\leq (D+1)^2, \end{aligned}$$

We substitute x by $\lambda y^\delta + z$. Therefore, $deg(\hat{Q}) \leq D(D+1) = \hat{D}$ and $max\text{-}coef(\hat{Q}) \leq \lambda^D 2^D M \leq 2^D M^{D+1} = \hat{M}$. By Using Case 4, we get a rational expression $Q'(y,z)$, and α_1', α_2' and π, such that for each $(x,y) \in S(1, \alpha_1', \alpha_2', \pi\alpha_3, \{\delta\}, \{\lambda\})$,

$$Q'(y,z) = \left\lfloor \hat{Q}(y,z) \right\rfloor = \lfloor P(x,y) \rfloor$$

where (y,z) is the (unique) generator of (x,y). Now substitute $z = x - \lambda y^\delta$ in $Q'(y,z)$ to get $Q(x,y)$. We compute the following bounds for Q,

$$\begin{aligned} \max\{deg(Q), \alpha_1'\} &\leq (\hat{D}+1)^2 \\ &\leq (D(D+1)+1)^2 \\ &\leq 4D^4, \end{aligned}$$

and

$$\begin{aligned} \max\{max\text{-}coef(Q), \alpha_2', \pi\alpha_3\} &\leq 2\left((\hat{D}+2)\hat{M}\right)^{\hat{D}+2} \\ &\leq 2\left((D(D+1)+2)(2^D M^{D+1})\right)^{D^2+2} \\ &\leq 2(4D^2 M)^{2D^3}. \end{aligned}$$

Since the relation between P_1 and P_2 belongs to one of the five cases, this concludes the proof of this lemma. □

The following lemma combines the previous lemmas into a proof technique for the two variable case. This lemma shows, that given a computation tree, one can find a subset, $S(r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda)$, such that all the inputs from this subset follow the same path. Once we establish this property, we can derive a lower bound for any boolean function that, for any such subset, has at least two elements which differ in their outcome. In the next section we show that deciding if two integers are relatively prime is one such function.

1. For each input $(a, b) \in \mathcal{S}^{(i)}$ the computation follows the path from the root to v_{i+1};

2. For each computation vertex ν on the path from the root to the vertex v_{i+1}, excluding the vertex v_{i+1}, there is a pair of bivariate polynomials $(F_\nu^i(x,y), G_\nu^i(x,y))$ with integer coefficients, such that for each input $(a,b) \in \mathcal{S}^{(i)}$, $G_\nu^i(u,v) \neq 0$, and $val(\nu) = \frac{F_\nu^i(u,v)}{G_\nu^i(u,v)}$, where (u,v) is the $< r^{(i)}, \alpha_1^{(i)}, \alpha_2^{(i)}, \alpha_3^{(i)}, \Delta^{(i)}, \Lambda^{(i)} >$-generator of (a,b); and

3. Let $\Sigma_i = \{F_{v_j}^i(x,y), G_{v_j}^i(x,y) \mid j \leq i\}$. Define $D_i = \max\{deg(\Sigma_i), 4\}$ and $M_i = \max\{max\text{-}coef(\Sigma_i), 4\}$. Then, $r^{(i)} \leq i$, $\mathsf{max}\{\alpha_1^{(i)}, D_i\} < 2^{2^{4i}}$, and $\mathsf{max}\{\alpha_2^{(i)}, \alpha_3^{(i)}, M_i\} < 2^{2^{2^{4i}}}$.

We construct the set $\mathcal{S}^{(i+1)} \subseteq \mathcal{S}^{(i)}$ such that Property 2 is satisfied also for the vertex v_{i+1} and each input $(a,b) \in \mathcal{S}^{(i+1)}$. We also select an outgoing edge of v_{i+1} and prove that for each input $(a,b) \in \mathcal{S}^{(i+1)}$ the computation follows this edge. It is easy to check that $r^{(i+1)} \leq i+1$. Therefore, in order to complete this proof, it is sufficient to show that the following two inequalities hold:

$$\begin{array}{lll} \text{(i)} & \mathsf{max}\{\alpha_1^{(i+1)}, D_{i+1}\} & < 2^{2^{4(i+1)}}, \\ \text{(ii)} & \mathsf{max}\{\alpha_2^{(i+1)}, \alpha_3^{(i+1)}, M_{i+1}\} & < 2^{2^{2^{4(i+1)}}}. \end{array}$$

Let us first resolve the case when v_{i+1} is a comparison vertex. Then by Lemma 4.3, we can extend $\mathcal{P}$. Furthermore, $\alpha_1^{(i+1)} \leq 2D_i + 1 < 2^{2^{4(i+1)}}$, and $\alpha_2^{(i+1)} \leq 2M_i^2(D_i+1)^2 + 1 < 2^{2^{2^{4(i+1)}}}$. Clearly, inequalities (i) and (ii) hold in this case.

Next, consider the case when v_{i+1} is a computation vertex. We divide the discussion to two cases. The first case is when the operation is a rational operation, and the second case is when the operation is floor.

Suppose v_{i+1} is $\nu \circ \mu$, where ν and μ are previous computation vertices, and $\circ \in \{+, -, *, /\}$. By the induction hypothesis, $val(\nu) = \frac{F_\nu(u,v)}{G_\nu(u,v)}$ and $val(\mu) = \frac{F_\mu(u,v)}{G_\mu(u,v)}$, where

$(a_0, a_1) = ((1+\alpha_3)^{(e+1)\alpha_1}, (1+\alpha_3)^e)$, and $(b_0, b_1) = (1 + \alpha_3(1+\alpha_3)^{(e+1)\alpha_1}, (1+\alpha_3)^e)$. In order to see that $(a_0, a_1) \in S(0, \alpha_1, \alpha_2, \alpha_3)$, one can verify that (1) $a_0 > a_1^{\alpha_1}$, (2) $a_1 = (1+\alpha_3)^e > \alpha_2$, and $a_0 \equiv a_1 \equiv 1 \bmod \alpha_3$. (A similar argument holds for (b_0, b_1).) Clearly, $\gcd(a_0, a_1) = (1+\alpha_3)^e \neq 1$, and $\gcd(b_0, b_1) = 1$.

In order complete this lemma, it is sufficient to prove each of the numbers a_0, a_1, b_0, and b_1 is less than $2^{2t(t+1)}$. Since e is the least exponent such that $(1+\alpha_3)^e > \alpha_2$, then $(1+\alpha_3)^e < 2^{2t}$. The desired upper bounds are an immediate consequence of this observation. □

Intuitively, the previous lemma shows, that if we terminate with a large subset $S(\cdot)$, there is still a pair of inputs, one of which is misclassified.

Theorem 4.8 *Any decision tree with $OP = \{+, -, *, /, \lfloor\cdot\rfloor\}$ and constants $\{0, 1\}$, that decides if a and b are relatively prime, for all integers $2^n > a > b > 0$, must have depth $\Omega(\log\log n)$.*

Proof: Without lost of generality we may assume that the leaves are labeled by the constants zero and one, and that the two first vertices are input vertices. (This can increase the depth of the tree by at most four.) Suppose that we are given a decision tree T of depth $h < \frac{1}{4}\log\log(n^{1/5})$, with $OP = \{+, -, *, /, \lfloor\cdot\rfloor\}$, that decides if a and b are relatively prime, for all integers $2^n > a > b > 0$. By Lemma 4.6 we have the following: (i) there is a path $\mathcal{P}$ from the root of T to a leaf ν, and a set of inputs $\mathcal{S} = S(r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda) \cap \{(a,b) : 0 < a, b < 2^n\}$, such that for each input $(a,b) \in \mathcal{S}$, the computation follows the path $\mathcal{P}$; (ii) The leaf ν is labeled by a constant (zero or one). (iii) $\alpha_1 < 2^{2^{4h}} < n^{1/5}$, α_2, $\alpha_3 < 2^{2^{2^{4h}}} < 2^{n^{1/5}}$, *max-coef*$(\frac{F_\nu(x,y)}{G_\nu(x,y)}) < 2^{n^{1/5}}$, and *deg*$(\frac{F_\nu(x,y)}{G_\nu(x,y)}) < n^{1/5}$. Our goal is to arrive at a contradiction using Lemma 4.7.

Towards this end, let $t = n^{1/5} - 1$. We claim that each pair $(u,v) \in S(0, \alpha_1, \alpha_2, \alpha_3) \cap \{(u,v) : 1 \leq u, v < 2^{2t(t+1)}\}$ generates a pair $(a,b) \in \mathcal{S} = \hat{S}(r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda) = S(r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda) \cap \{(a,b) : 0 < u, v < 2^n\}$. Recall that the first two vertices of the

Chapter 5

Approximation of Real Functions

5.1 Motivation

It is not hard to show that the square root function cannot be computed using rational operations only. For example $\sqrt{2}$ is not a rational number; therefore it cannot be represented by rational operations on $\{0, 1, 2\}$. The floor operation would not resolve this problem, since its output is always an integer. Still, we would like to argue about computations that involve such functions.

One direction of research may be to assume that the square root function is one of our basic operations in the computation tree, and consider the complexity of computing other functions, given it as a basic operation. Another direction is to relax the requirement on the output of the computation tree. Instead of requiring it to be equal to the computed function, we can require it to "approximate" the computed function.

The first approach, where the square root is consider as a basic operation, was investigated in [SY76, Pip81, Ja'81]. The main complexity measure that they considered was the number of square root operations that have to be performed. It was shown, for

The norm L_∞ is defined as,

$$||f||_\infty = \sup_{x \in \Lambda} |f(x)|$$

It is well known that the norm of a function $||f||_k$ is monotonically non-decreasing in the parameter k, i.e. $||f||_1 \leq ||f||_2 \leq \cdots \leq ||f||_\infty$. This implies that showing a lower bound for $||f||_1$, implies a lower bound for any L_k, including L_∞.

We still need to make a connection between the norm of a function and the quality of an approximation. Given a certain point x, the difference between $g(x)$ and $h(x)$ is a measure of how well g approximates h at that point. This suggests that the norm of $g - h$ would be a good indicator of how well g approximates h. More formally, we define a function $error_k(g, h)$ as follows,

$$error_k(g,h) = ||g-h||_k = \sqrt[k]{\int_\Lambda |g(x) - h(x)|^k dx}$$

The function $error_k(g, h)$ is how well g approximates h in norm L_k. A function $g(x)$ approximates a function $h(x)$, within ϵ, in norm L_k, if $error(g, h)_k \leq \epsilon$.

The more intuitive norms are probably L_∞ and L_1 (although norm L_2 plays an important role in many applications). The norm L_∞ is intuitively the worst case, for approximation it would mean the worst case approximation, over any input. An ϵ approximation in norm L_∞ implies that for *any* input in Λ, the difference between the two functions is at most ϵ. The other norm is L_1, which intuitively captures the *average* case. In the case of approximation one can view it as the expected error for a randomly chosen input.

In this chapter we show a technique to derive lower bounds for approximation in norm L_1. Clearly any lower bound in norm L_1, implies a lower bound in any other norm, including L_∞.

path, $\mathcal{P}$, in the computation tree. Note that this would not be true for an integer input, since, for example, branching according to whether the input is odd or even, divides the integers to two sets, such that neither set has two successive integers. The other point is that we need l to be "large", otherwise we will not be able to establish a lower bound at the end. For example, if $l \leq \epsilon$, then the square root function can be approximated by a constant over such an interval.

The proof that there exists a path $\mathcal{P}$ is done by constructing it inductively. The inductive claim uses a prefix $\mathcal{P}_i$ of $\mathcal{P}$, that includes the first i vertices, and a subinterval $[a_i, a_i + l_i]$, such that the value of each vertex in $\mathcal{P}_i$, for inputs from $[a_i, a_i + l_i]$, can be expressed as a rational expression, and any input from $[a_i, a_i + l_i]$ follows $\mathcal{P}_i$.

In the inductive step we show that $\mathcal{P}_i$ can be extended to $\mathcal{P}_{i+1}$ and maintain the inductive claim. Here we have two cases, depending on the type of last vertex of $\mathcal{P}_i$.

If the last vertex in $\mathcal{P}_i$ is a comparison vertex, we need to show how to choose one of its sons as the next vertex v_{i+1}. In this case we need to show that there is a "large" subinterval $[a_{i+1}, a_{i+1} + l_{i+1}]$, where all the inputs continue to v_{i+1}.

The other case is when the next vertex on $\mathcal{P}$ is a computation vertex. If the computation vertex has a rational operation, then the inductive claim is maintained trivially. The interesting case is when the vertex computes the floor function. Such a vertex receives as an argument the value of some previous vertex on $\mathcal{P}_i$, which by the induction hypothesis can be expressed as a rational expression. Using the above argument about $\lfloor r(x) \rfloor$, there is a "large" subinterval $[a_{i+1}, a_{i+1} + l_{i+1}]$, in which the value of this vertex is constant. We continue by setting the value of the vertex to that constant, and restrict the inputs to the subinterval $[a_{i+1}, a_{i+1} + l_{i+1}]$.

To summarize, the construction so far shows that for any computation tree T, defined over inputs $x \in [1,2]$, there is a subinterval $[a, a + l]$, and a rational expression $r(x)$, such that $T(x) = r(x)$ for any $x \in [a, a + l]$. This by itself does not imply any lower bound but reduces the problem of proving the lower bound for T, over $[1,2]$, to the problem of lemma shows that the "penalty" of restricting the rational function to a subinterval in which it has no roots or poles is rather limited.

Lemma 5.2 *Given an interval $[a, a+l]$ and a rational function $r(x)$ of degree d, there exists a subinterval $[\alpha, \alpha + \frac{l}{7d}] \subset [a, a+l]$ that does not contain any poles or zeros of $r(x)$, and furthermore, $r(x)$ is monotone in this subinterval.*

Proof: Let $r(x) = \frac{p(x)}{q(x)}$ where $p(x)$ and $q(x)$ are polynomial of degree at most d. Consider the $6d+1$ subintervals $[a, a+\frac{l}{6d+1}], [a+\frac{l}{6d+1}, a+\frac{2l}{6d+1}], \ldots, [a+\frac{6dl}{6d+1}, a+l]$. In each subinterval in which $r(x)$ has a pole, the polynomial $q(x)$ has a zero. In the worst case, each zero of $q(x)$ can be at the boundary of two adjacent subintervals. Since $q(x)$ has at most d zeros, the number of subintervals that contain a pole of $r(x)$ is at most $2d$. Similarly, the number of intervals that contain a zero of $r(x)$ (which is a zero of $p(x)$) is bounded by $2d$. Therefore, there are $k \geq 2d+1$ intervals that do not contain any pole or zero of $r(x)$.

Now, consider these remaining k intervals. Each interval where $r(x)$ is not monotone contains either a local maxima or a local minima in its interior. At each local maxima or local minima, the derivative of $r'(x)$ has a zero. Since the degree of $r'(x)$ is at most $2d$, it can have at most $2d$ zeros. (Notice that the poles of $r'(x)$ are same as the poles of $r(x)$.) Hence, there exist $k - 2d \geq 1$ subintervals that do not contain any poles or zeros of $r(x)$, and furthermore, $r(x)$ is monotone in each of these subintervals. □

The following lemma is a classical result, known as the Markoff inequality. It relates the value of the derivative of a polynomial to the polynomial value and degree.

Lemma 5.3 (Markoff [Che66, page 91]) *Let $p(x)$ be a polynomial of degree d, and $p'(x)$ the derivative of $p(x)$ with respect to x. Then*

$$\max_{x\in[-1,1]}\{|p'(x)|\} \leq d^2 \max_{x\in[-1,1]}\{|p(x)|\}.$$

on polynomials, we can also bound the value of a rational expressions. Using the above lemma, we can derive the following lower bound for a polynomial.

Lemma 5.6 *Given an interval* $[a, a+l]$ *and a polynomial* $p(x)$ *of degree* $d \geq 1$ *with* integer *coefficients, there exists a subinterval* $[\alpha, \alpha + \frac{l}{8d^3}] \subset [a, a+l]$ *such that (i)* $p(x)$ *is monotone in this subinterval, and (ii)* $|p(x)| \geq (\frac{l}{4})^d$ *for all* $x \in [\alpha, \alpha + \frac{l}{8d^3}]$.

Proof: Let b be the leading coefficient of p. Since $d \geq 1$ then $b \neq 0$, furthermore, since b is an integer, this implies that $|b| \geq 1$. Assume that K is the largest value of $|p(x)|$ in the interval $[a, a+l]$. First, we argue that there is a subinterval of length $\delta = \frac{l}{8d^3}$ such that $p(x)$ is monotone and is greater than $K/2$, throughout this subinterval. Then, we show that $K/2 \geq (l/4)^d$.

Let $y_i = a + \frac{il}{8d^3}$ for $i = 0, 1, \ldots, 8d^3$. Consider the $8d^3$ subintervals $[y_{i-1}, y_i]$, for $i = 1, 2, \ldots, 8d^3$. Choose $x_0 \in [a, a+l]$ such that $|p(x_0)| = K$. Choose j such that $x_0 \in [y_j, y_{j+1}]$. Assume that $j \leq 4d^3$, and consider d consecutive subintervals $[y_{j+i}, y_{j+i+1}]$ for $i = 0, 1, \ldots, d-1$. (The case $j \geq 4d^3$ is similar.) Since the polynomial $p(x)$ has a total of at most $d-1$ local maxima and local minima, $p(x)$ is monotone in one of these d subintervals.

Next, we show that $|p(x)| \geq K/2$ for all $x \in [y_j, y_{j+d}]$. For all real u such that $x_0 + u \in [a, a+l]$, Lemma 5.4 implies that $|p(x_0 + u)| \geq |p(x_0)| - u\frac{2d^2}{l}K$. Therefore, for all $x \in [y_j, y_{j+d}]$, $|p(x)| \geq K - \frac{l}{8d^2}\frac{2d^2}{l}K = K(1 - \frac{1}{4}) > \frac{1}{2}K$.

It remains to show now that $K/2$ is larger than $(\frac{l}{4})^d$. In order to prove this assertion, transform the polynomial $p(x)$ into a polynomial $q(z)$ with leading coefficient one, in the interval $[-1, 1]$ by setting

$$q(z) = \frac{1}{b}(\frac{2}{l})^d p(z\frac{l}{2} + \frac{2a+l}{2}), \text{ where } b \text{ is the leading coefficient of } p(x).$$

exists a subinterval $[\alpha, \alpha + \frac{l}{56d^4}]$, such that $|r(x)| \geq (2M)^{-1}(\frac{8}{l})^{-d}$ for all $x \in [\alpha, \alpha + \frac{l}{56d^4}]$, and $r(x)$ is monotone in this subinterval.

The following lemma is the main technique that we use to bound the value of the floor function. This lemma guarantees that there is a "large" subinterval in which the value of the floor function is constant. This implies that if we restrict our attention only to this subinterval, the value of the floor operation is constant.

Lemma 5.9 *Given an interval $[a, a + l]$, where $1 \leq a \leq 2$ and $0 < l \leq 2 - a$, and a rational function $r(x)$ of degree d with* integer *coefficients and max-coef$(r) \leq M$; there exists α such that $\lfloor r(x) \rfloor$ equals a fixed integer J for all $x \in [\alpha, \alpha + M^{-1}(\mu l)^{d+1}] \subset [a, a + l]$, where $\mu < 1$ is a fixed constant that can be chosen to be* 0.02. *Moreover, $|J| \leq M(\mu l)^{-d}$.*

Proof: By Lemma 5.7, there exists a subinterval $[\beta, \beta + \frac{l}{56d^4}]$ in which $r(x)$ is monotone, and $|r(x)| \leq 2M(\frac{8}{l})^d$ for all $x \in [\beta, \beta + \frac{l}{56d^4}]$. This implies that $\lfloor r(x) \rfloor$ takes on at most $4M(\frac{8}{l})^d$ distinct integer values in this subinterval. By the pigeon-hole principle, there is a subinterval of length at least $\frac{l}{56d^4 4M(\frac{8}{l})^d}$ where $r(x)$ is a constant. Let J equal the value of $r(x)$ in that interval. □

So far we were interested in the behavior of rational functions and the floor function. The next lemma shows a property of the s^{th} root function. This lemma establishes a connection between the degree of a rational expression, its maximum coefficient, and how well it approximates the s^{th} root.

We should note that one can bound the approximation as a function of the degree, independent of the size of the maximum coefficient. The techniques used to achieve such bounds are more involved, and are not needed in our case. (See, for example, [Bra86].)

Lemma 5.10 *For an integer $s \geq 2$, given an interval $[a, a+l] \subset [1, 2]$ where $1 \leq a < 2$ and $0 < l \leq 2 - a$, and a rational function $r(x)$ of degree d with integer coefficients and*

2. For each computation vertex ν on the path $\mathcal{P}$, there is a rational function $r_\nu(x)$, with integer coefficients, such that for any input $\alpha \in [a, a+l]$, the value computed at vertex ν equals $r_\nu(\alpha)$.

Finally, we conclude the desired lower bound on the length of $\mathcal{P}$ (and hence, a lower bound on the depth of T) by applying the result of Lemma 5.10 to rational functions that approximate $\sqrt[s]{x}$ over the interval $[a, a+l]$.

Denote the vertices on the path $\mathcal{P}$ from the root of T to its leaf v_l by $v_1, v_2, \ldots, v_l$, in that order, where v_1 is the root of the tree T and v_i is a child of v_{i-1}. We define the path $\mathcal{P}$ and the subinterval inductively, starting with the empty path and the interval $[1, 2]$. As part of the induction hypothesis, we maintain three properties of the path and the interval under consideration. These properties are described below.

Suppose that (a) we have selected a prefix of $\mathcal{P}$, which starts at v_1, and ends at a vertex v_{i+1}, and (b) defined a subinterval $[a_i, a_i + l_i]$ with the following properties:

1. For each input $x \in [a_i, a_i + l_i]$ the computation follows the path from the root to v_{i+1};

2. For each computation vertex v_j on the path from the root to the vertex v_{i+1}, excluding the vertex v_{i+1}, there is a rational function $r_j(x)$, with integer coefficients, such that $val(v_j) = r_j(x)$, for any input $x \in [a_i, a_i + l_i]$,

3. Let $\Sigma_i = \{r_j(x) | v_j \text{ is a computation vertex, } j \leq i\}$. Define $D_i = deg(\Sigma_i) + 1$ and $M_i = max\text{-}coef(\Sigma_i)$. Then, $D_i \leq 2^i$, $M_i \leq \mu^{-2^{2i^2}}$, and $l_i \geq \mu^{2^{2i^2}}$, where μ is the constant in the statement of Lemma 5.9.

The basis of the induction is $i = 0$, and the path is v_1, which is the root. All the claims hold trivially in this case.

We show how to define $[a_{i+1}, a_{i+1} + l_{i+1}]$ and how to choose the vertex v_{i+2} such that Properties 1-3 will be satisfied by the interval $[a_{i+1}, a_{i+1} + l_{i+1}]$ and the prefix of $\mathcal{P}$ that bounded by $2(D_i + 1)M_i^2$. Therefore, $D_{i+1} < 2D_i$, $M_{i+1} \leq 2(D_i + 1)M_i^2$. For each $x \in [a_{i+1}, a_{i+1} + l_{i+1}]$ and $r_{i+1}(x)$, Properties 1-3 are satisfied.

Suppose that $f_{v_{i+1}} = \lfloor f_{v_j} \rfloor = \lfloor g(x) \rfloor$, for some $1 \leq j \leq i$. By our hypothesis, *deg*$(g) < D_i$ and *max-coef*$(g) \leq M_i$. By Lemma 5.9, there exists an integer J, and an interval $[a_{i+1}, a_{i+1} + l_{i+1}]$, such that (i) for any $x \in [a_{i+1}, a_{i+1} + l_{i+1}]$, $\lfloor g(x) \rfloor = J$, $|J| \leq M_i(\mu l_i)^{-D_i} \leq \mu^{-2^{2i^2}} \mu^{-2^i(1+2^{2i^2})} \leq \mu^{-2^{2(i+1)^2}}$; and (ii) $l_{i+1} \geq \frac{1}{M_i}(\mu l_i)^{D_i+1} \geq \mu^{2^{2i^2}} \mu^{(1+2^{2i^2})(2^i+1)} \geq \mu^{2^{2(i+1)^2}}$. Define $r_{i+1}(x) = J$. Then, $D_{i+1} = D_i$ and $M_{i+1} = \max\{M_i, J\}$, and properties 1-3 are satisfied.

Thus, we have proved the existence of a path $\mathcal{P}$ and an interval $[a, a + l]$ such that properties 1-3 are satisfied for all vertices on $\mathcal{P}$. We have also established that there is a rational function $r(x)$ such that for all $\alpha \in [a, a + l]$ the value produced by T (at the end of the path $\mathcal{P}$) on input α is given by $r(\alpha)$. Furthermore, that *deg*$(r) < 2^h$, $M =$ *max-coef*$(r) \leq \mu^{-2^{2h^2}}$, and $l \geq \mu^{2^{2h^2}}$.

By Lemma 5.10 there exists a subinterval $[\alpha, \alpha + l']$, $l' = \frac{l}{4(56s(2sd+2)^4)} = \Omega(\frac{l}{s^5 d^4})$, such that for any v in the subinterval, $|v^{1/s} - r(v)| \geq c(M(sd + 2))^{-4}(\frac{l}{8s})^{2sd+3} = \delta$, where c is a constant. This implies that we can lower bound $error_1(T, \sqrt[s]{x})$ as follows,

$$error_1(T, \sqrt[s]{x}) \geq \delta l'$$

The value of δ is $\Omega(M^{-4}(\frac{l}{s})^{O(sd)})$, and we can show $\delta \geq c_s^{2^{4h^2}}$, where c_s is a constant that depends only on s. Since T approximates $\sqrt[s]{x}$, in L_1, within ϵ, then

$$\epsilon \geq error_1(T, \sqrt[s]{x}) \geq \delta l' \geq c_s^{2^{5h^2}}$$

Therefore, $h = \Omega(\sqrt{\log \log \frac{1}{\epsilon}})$. □

Chapter 6

Upper Bounds

In this chapter we show how the power of the floor function can be used to accelerate computations. Clearly the floor function can help in computing the floor function and the mod function, but the interesting fact is that it can accelerate computations which we normally do not associate with the floor function.

One issue of interest is the constants that the computation tree uses. A *uniform* computation tree has as constants only $\{0, 1\}$, and every other constant has to be generated explicitly. On the other hand a *non-uniform* computation tree may have arbitrary initial constants.

We show that many functions can be computed by non-uniform computation trees of depth $O(1)$. This justifies the reason that the lower bounds proved in the previous chapters are for the uniform computation trees. For uniform computation trees we show upper bounds that, in some cases, match the lower bound that we proved in the previous chapters.

We start, in Section 6.1, by showing that any polynomial has a non-uniform computation tree of depth $O(1)$ that evaluates it, and this constant does not depend on the degree

First we show how to construct a constant that encodes all the coefficients of a polynomial $P(y)$. The constant has simply all the coefficients in order, each padded to m bits, where m will be chosen latter. More formally, let $P(y) = \sum_{i=0}^{d} p_i y^i$, where the coefficients, p_i, are integers. Define $c_p^m = P(2^m) = \sum_{i=0}^{d} p_i 2^{mi}$. For m, such that $2^m > \textit{max-coef}(P)$, it is easy to verify that the m-bit integer obtained by considering bit positions im to $(i+1)m-1$ of c_p^m is equal to p_i.

The polynomial evaluation algorithm:

The algorithm is constructed to evaluate the inputs of a specific polynomial P in constant number of operations. The algorithm receives as an *input* an n-bit integer a. The algorithm *output* is the value of $P(a)$. The polynomial evaluation algorithm consists of two steps.

Step 1. Compute $b = \left\lfloor \frac{2^{md}}{1-2^{-m}a} \right\rfloor$. Letting $\gamma(a) = \sum_{i=0}^{d} a^i 2^{m(d-i)}$, we show that $b = \gamma(a)$. For $|x| < 1$, the following identity holds, $\frac{1}{1-x} = \sum_{i=0}^{\infty} x^i$. Recall that $a < 2^n$. We will chose m such that $2^n < 2^m$, and therefore $2^{-m}a < 1$. This implies that $\frac{1}{1-2^{-m}a} = \sum_{i=0}^{\infty}(2^{-m}a)^i$, and therefore

$$\frac{2^{md}}{1-2^{-m}a} = \sum_{i=0}^{\infty} 2^{md}(2^{-m}a)^i = \sum_{i=0}^{\infty} 2^{m(d-i)}a^i.$$

For m, such that $a^{d+1}/2^m < 1/2$, the sum $\sum_{i=d+1}^{\infty} 2^{m(d-i)}a^i$ is less than one. This, in combination with the fact that a is an integer, implies that

$$b = \left\lfloor \sum_{i=0}^{\infty} 2^{m(d-i)}a^i \right\rfloor = \sum_{i=0}^{d} 2^{m(d-i)}a^i + \left\lfloor \sum_{i=d+1}^{\infty} 2^{m(d-i)}a^i \right\rfloor = \gamma(a).$$

The constants used in this step are 2^{md} and 2^m, and the requirement on m is that $m \geq n(d+1)+1$.

Step 2. Extract the m-bit integer obtained by considering bit positions md to $m(d+1)-1$

6.2 Approximations

The non-uniform case

We show how to use the construction in the previous section, to approximate $\sqrt{b}$ for any $b \in [1,2]$ in constant time. The existence of such a program does not contradict the lower bounds of Chapter 5, because it assumes that certain large constants are available for "free". In other words, we show a $O(1)$ non-uniform upper bound on the depth of computation trees for approximating $\sqrt{x}$.

For $b \in [1,2]$, let $a = \lfloor 4b/\epsilon^2 \rfloor$. It is not difficult to see that if $|\sqrt{a} - \alpha| < 1$ then $|\sqrt{b} - \alpha\epsilon/2| < \epsilon$. Below, we present an $O(1)$ step algorithm for computing α.

Our algorithm is based on Newton iteration for computing square roots. Recall that Newton iteration for computing the square root of a is given by

$$x_{i+1} = \frac{a + x_i^2}{2x_i}, \text{ where } x_0 \text{ is the starting point.}$$

Define δ_i to be the relative error of x_i, i.e. $(1+\delta_i)\sqrt{a} = x_i$. It is easy to verify that

$$\delta_i = \frac{\delta_{i-1}^2}{2(1+\delta_{i-1})} \leq 2(\frac{\delta_0}{2})^{2^i}, \text{ assuming that } \delta_0 < 1$$

Let $x_0 = \lfloor 3/\epsilon \rfloor$, this implies that $\sqrt{a} \leq x_0 \leq \frac{3}{2}\sqrt{a}$. Since $0 \leq \delta_0 \leq 1/2$, this implies that $\delta_i \leq 2 \cdot 4^{-2^i}$. Our aim is to reach an index t, such that $|\sqrt{a} - x_t| < 1$. This occurs when $|\delta_t\sqrt{a}| < 1$, or equivalently $|\delta_t| < \frac{1}{\sqrt{a}}$. For $t = O(\log\log a) = O(\log\log(1/\epsilon))$, the value of δ_t satisfies, $0 \leq \delta_t < 1/\sqrt{a}$. Thus, in order to compute α, it is sufficient to perform t Newton iterations, starting at x_0.

Notice that upon starting the Newton iterations at x_0, the value of x_1 is obtained by evaluating the rational function $R_1(y) = \frac{P_1(y)}{Q_1(y)}$ at $y = a$, where $P_1(y) = y + x_0^2$ and

The uniform case

The general idea in this case is similar to the idea in the non-uniform case. Using the same logic as before, for any $b \in [1,2]$ and $a = \lfloor 4b/\epsilon^2 \rfloor$, if $|\sqrt{a} - \alpha| < 1$ then $|\sqrt{b} - \alpha\epsilon/2| < \epsilon$.

The main difference between the construction in the uniform case and the non-uniform case is the following. In the uniform case we will consider the approximation, x_i, computed at the i^{th} Newton iteration, as a function of the starting point x_0, while assuming that a is constant. Recall that for the non-uniform case the roles were reversed; x_i was viewed as a function of a, while x_0 was fixed.

For a starting point x_0, the value of the i^{th} approximation, x_i, is given by evaluating a rational function $H_i(x) = \frac{F_i(x)}{G_i(x)}$ at the point $x = x_0$, where $F_1(x) = x^2 + a$, $G_1(x) = 2x$; and for $i > 1$, $F_i(x) = aG_{i-1}^2 + F_{i-1}^2$, $G_i(x) = 2F_{i-1}(x)G_{i-1}(x)$. (Note that the coefficients of F_i and G_i are integers.)

The above identities can be used to define a straight line program, of length $O(t)$ that computes H_t. The rational expression H_t performs t Newton iteration for a certain input a. This implies that $H_t(H_t(x_0))$ can be viewed as first performing t iterations, starting at x_0 and ending at $x_t = H_t(x_0)$, and then performing t more iterations, starting at x_t and ending at $x_{2t} = H_t(x_t)$; and therefore, $H_t(H_t(x)) = H_{2t}(x)$.

For the sake of simplicity we assume that ϵ is a power of two[1]. Let $d_t = \deg(H_t)$, $M_t = \textit{max-coef}(H_t)$, and $2^n = \frac{8}{\epsilon^2}$. Initially, $\textit{max-coef}(F_1)$, $\textit{max-coef}(G_1) \leq \frac{8}{\epsilon^2} = M_0$. From the definition of F_i and G_i it is clear that $\deg(F_i), \deg(G_i) \leq 2^i$. One can verify that $\textit{max-coef}(F_i)$, $\textit{max-coef}(G_i) \leq (M_0+1)^{2^{2i}}$. Therefore, $d_t = 2^t = d$ and $M_t \leq (M_0+1)^{2^{2t}} \leq (\frac{16}{\epsilon^2})^{2^{2t}} = M$.

We choose m, such that $2^m = 2(\frac{16}{\epsilon^2})^{2^{2t+1}} \geq 2M2^{2nd}$. Clearly 2^m and 2^{md} can be computed in $O(t)$ steps. Since H_t has a straight line program of length $O(t)$, and we

[1] For an arbitrary ϵ we can use the powers of two subroutine, describe in the next section, to find an $\epsilon' < \epsilon$ which is a power of two. The running time of this procedure would be $O(\sqrt{\log\log\frac{1}{\epsilon}})$.

```
input:        $b, \epsilon$
computation:
              Compute $2^m$, $2^{dm}$ and $c_{H_t}^m = H_t(2^m)$.
              $\hat{x_0} = \left\lfloor \frac{4b}{\epsilon^2} \right\rfloor$
              FOR $i = 1tok$ DO
                  $\hat{x_i} = \lfloor H_t(\hat{x_{i-1}}) \rfloor$/* invoke the polynomial evaluation procedure */
              $\alpha = \hat{x_k} - 1$
              $\beta = \frac{\alpha\epsilon}{2}$
output $\beta$.
```

Figure 6-2: Approximating the square root uniformly

would be $\alpha\epsilon/2$. To conclude we proved the following theorem.

Theorem 6.3 *There exists a computation tree $T(\epsilon, x)$, whose depth is $O(\sqrt{\log\log\frac{1}{\epsilon}})$, such that for any $b \in [1, 2]$,*

$$|T(\epsilon, b) - \sqrt{b}| \leq \epsilon$$

Note that the algorithm receives two inputs, ϵ and b, and outputs $T(\epsilon, b)$.

Approximating the square root of an integer is closely related to deciding if an integer is a perfect square (i.e. whether an integer has an integral square root). We could decide if an n bit integer is a perfect square by first approximating its square root within one, and then checking the integers within one of that value. Note that in the approximation procedure above, α approximates $\sqrt{a}$ within one, where a is an integer. The problem in extending the result to deciding perfect squares is that we need to compute a "good" initial point, (i.e. such that $0 \leq \delta_0 \leq 1/2$) and a bound on M_0. For the approximation case, we chose the value of a such that it would be easy to compute a "good" initial point for it. The following lemma states that given a "good" initial point, and a constant $2^l > a$, we can decide the perfect square problem in $O(\sqrt{\log\log a})$.

Lemma 6.4 *There exists a computation tree $T(x, x_0, 2^l)$, whose depth is $O(\sqrt{\log\log x})$,* computing $b_i = a^k \bmod (a - b_{i-1})$, we can raise b to the ik power in $O(i)$ steps, as long as $b^{ik} < a - b^{(i-1)k}$.

For a given an integer a, we show how to compute $2^{2^{i^2}}$ for all i, such that $2^{2^{i^2}} < a$. The computation is done in i steps. Starting with $a_1 = (2a)^2$ and $b_1 = 2^2$ we compute: $a_i = a_{i-1}^4$ and $b_i = a_i \ \mathsf{mod}\ (2a - b_{i-1})$. (Recall that the **floor** and the **mod** operations are computationally equivalent, i.e. mod can be computed in $O(1)$ operations using the **floor** operation.)

We prove the correctness of the computation by induction. It is easy to see that $a_i = (2a)^{2^{2i-1}}$. Recall that $x - y$ divides $x^k - y^k$, for any positive integer k, therefore $2a - b_{i-1}$ divides $(2a)^k - b_{i-1}^k$, for $k = 2^{2i-1}$. Thus,

$$\begin{aligned} b_i &\equiv a_i \mathsf{mod}\ (2a - b_{i-1}) \\ &\equiv (2a)^{2^{2i-1}} - \left((2a)^{2^{2i-1}} - b_{i-1}^{2^{2i-1}}\right) \mathsf{mod}\ (2a - b_{i-1}) \\ &\equiv b_{i-1}^{2^{2i-1}} \mathsf{mod}\ (2a - b_{i-1}). \end{aligned}$$

Observe that $b_{i-1}^{2^{2i-1}} = 2^{2^{i^2}}$. We get that if $b_{i-1}^{2^{2i-1}} < 2a - b_{i-1}$ then $b_i = 2^{2^{i^2}}$. By the induction hypothesis $b_{i-1} = 2^{2^{(i-1)^2}} < a$. Thus, if $2^{2^{i^2}} < a$, then $2^{2^{i^2}} < 2a - b_{i-1}$. Clearly the computation is done in $O(i)$ steps.

Seemingly, the above procedure gives us a way to compute $2^{2^{i^2}}$ for $i = 1, \ldots,$ $\left\lfloor\sqrt{\log\log a}\right\rfloor$. However, note that $\left\lfloor\sqrt{\log\log a}\right\rfloor$ is not known in advance. Thus, we are left with the following problem: How can we identify when to stop? That is, how to find the first i for which $2^{2^{i^2}} > a$ and thus $b_i \neq 2^{2^{i^2}}$. We cannot spend the time testing whether $b_i = b_{i-1}^{2^{2i-1}}$ by successive squaring, for each i, since it requires too many steps.

The solution is to compute another variable d_i, such that $d_i = 2^{2^{i^2}}$, for $i = 1, \ldots, l+1$, where l is the greatest integer such that $b_l = 2^{2^{l^2}}$. This implies that $d_i = b_i$ for $i \leq l$ and $d_{l+1} \neq b_{l+1}$. This allows us to detect the termination by simply comparing b_i to d_i and

Applications

We can now show a few applications of the powers of two subroutine. The first application is computing $\left\lfloor\sqrt{\log\log a}\right\rfloor$. Let t be the last iteration in which $b_t = d_t$. Then,

$$\left\lfloor\sqrt{\log\log a}\right\rfloor = \begin{cases} t-1 & \text{if } b_t > a \\ t & \text{if } b_t \le a \end{cases}$$

Thus, we can compute $\left\lfloor\sqrt{\log\log a}\right\rfloor$ in $O(\sqrt{\log n})$ time, for all n-bit integers.

We extend the result of computing $\left\lfloor\sqrt{\log\log x}\right\rfloor$ and show how to compute $\lfloor\log\log x\rfloor$. Using the powers of two procedure we can compute in $O(t)$ steps, the number $2^{2^{t^2}}$, such that $2^{2^{t^2}} \le a < 2^{2^{(t+1)^2}}$. The idea is that in $O(t)$ additional steps, we can find a j such that $2^{2^j} \le a < 2^{2^{j+1}}$.

This is done by simply squaring $2^{2^{t^2}}$ successively, until the first time it is larger than a. Note that since $(t+1)^2 - t^2 = 2t-1$, at most $2t-1$ successive squaring will be performed. We conclude that given an n-bit input a we can compute $j = \lfloor\log\log a\rfloor$ in $O(\sqrt{\log\log a}) = O(\sqrt{\log n})$ time. In Chapter 3 we proved an $\Omega(\sqrt{\log n})$ time lower bound for the respective decision problem. Therefore we have proved the following tight bounds.

Theorem 6.5 *There is a decision tree with* $OP = \{+, -, *, /, \lfloor\cdot\rfloor\}$ *and constants* $\{0,1\}$ *that computes* $\lfloor\log\log a\rfloor$*, for all n-bit integers a, and has depth* $O(\sqrt{\log n})$*. Furthermore, any decision tree for this problem has depth* $\Omega(\sqrt{\log n})$*.*

We can also use the powers of two subroutine to show that for infinitely many n, there is a decision tree, of depth $O(\sqrt{\log n})$, that decides if an n-bit integer is a perfect square. In this case we restrict the integer input to be from the interval $[2^{n-1}, 2^n)$. Suppose that $n = 2^{t^2+1}$. In $O(\sqrt{\log n})$ time we can compute $2^{2^{t^2}} = 2^{n/2}$, and $2^{2^{(t+1)^2}} > 2^n$, starting from the constants zero and one, and the input. Using Lemma 6.4, with the initial point

Chapter 7

Conclusions

In this Chapter we give a brief overview of the results that were presented in this thesis. We also mention a few open problems and possible directions for future research.

In Chapter 3 we developed a lower bound technique for decision tree with operations $\{+, -, *, /, \lfloor\cdot\rfloor\}$, This lower bound technique could be used to prove $\Omega(\sqrt{\log n})$ lower bounds for problems, such as deciding perfect squares, computing $\lfloor \log\log x \rfloor$, and other problems.

As a historical remark, it is interesting that when the lower bounds were proven, there was a gap between the lower bound of $\Omega(\sqrt{\log n})$ and the upper bounds of $O(\log n)$. At that time it was believed that the gap could be closed by improving the lower bounds. Only later, when considering closely the lower bound argument to see where it could be tightened, did the technique of achieving the upper bound emerge. This, and the other upper bounds in Chapter 6, show that the floor operation can add significant power to the computation.

In Chapter 4 we have proved an $\Omega(\log\log n)$ lower bound on the depth of any computation tree with operations from the set $\{+, -, *, /, \lfloor\cdot\rfloor\}$, that decides if two n-bit integers

Bibliography

[Bas90a] Nader Bashouty. Euclide's GCD algorithm is not optimal. Manuscript, 1990.

[Bas90b] Nader Bashouty. $\Omega(\log\log(1/\epsilon))$ lower bound for approximating the square root. Manuscript, 1990.

[BJM88] L. Babai, B. Just, and F. Meyer auf der Heide. On the limits of computations with the floor function. *Information and Computation*, 78(2):99–107, August 1988.

[BMS81] A. Bertoni, G. Mauri, and N. Sabadini. A characterization of the class of functions computable in polynomial time on random access machines. In *Proc. of the 13th ACM Symp. on Theory of Computing*, pages 168–176, Milwaukee, May 1981.

[BO83] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. of the 15th ACM Symp. on Theory of Computing*, pages 80–86, Boston, May 1983.

[Bra86] D. Braess. *Nonlinear Approximation Theory.* Springer-Verlag, Berlin, 1986.

[CG90] B. Chor and O. Goldreich. An improved parallel algorithm for integer gcd. *Algorithmica*, 1990. to appear.

[Che66] E. W. Cheney. *Introduction to Approximation Theory.* McGraw-Hill, New York, NY, 1966.

on Foundations of Computer Science, Singer Island, Florida, pages 7–11, October 1984.

[Knu81] D.E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, Ma, second edition, 1981.

[MST88] Y. Mansour, B. Schieber, and P. Tiwari. Lower bounds for integer greatest common divisor computations. In 29^{th} *Annual Symposium on Foundations of Computer Science, White Plains, New York*, pages 54–63, 1988. to appear in JACM.

[MST89a] Y. Mansour, B. Schieber, and P. Tiwari. The complexity of approximating the square root. In 30^{th} *Annual Symposium on Foundations of Computer Science, Research Triangle Park, NC*, pages 325–330, 1989.

[MST89b] Y. Mansour, B. Schieber, and P. Tiwari. Lower bounds for computations with the floor operation. In *Proceeding of ICALP*, 1989.

[Pip81] N. Pippenger. Computational complexity of algebraic functions. *Journal of Computer and System Science*, 22:454–470, 1981. See also, Correction to: Computational Complexity of Algebraic Functions. *Journal of Computer and System Science*, 37:395-399, 1988.

[PS76] V.R. Pratt and L.J. Stockmeyer. A characterization of the power of vector machines. *Journal of Comput. and Sys. Sci.*, 12:198–221, 1976.

[Rab72] M. Rabin. Proving the simultaneous positivity of linear forms. *J. Comp. and Syst. Sci.*, 6(6):639–650, 1972.

[Rei71] E. Reingold. Computing the maximum and the median. In *12th Annual Symposium on Switching and Automata Theory*, pages 216–218, 1971.

[Riv69] T.J. Rivlin. *An Introduction to the Approximation of Functions.* Dover Publications, New York, NY, 1969.

[Sha79] A. Shamir. Factoring numbers in $O(\log n)$ arithmetic steps. *Info. Proc. Letters*, 8(1):28–31, January 1979.

[Sim81] J. Simon. Division in idealized unit cost RAMs. *Journal of Comput. and Sys. Sci.*, 22:421–441, 1981.

[Sto76] L. Stockmeyer. Arithmetic versus boolean operations in idealized register machines. Technical Report RC 5954, IBM T.J. Watson Research Center, Yorktown Heights, April 1976.

[SY76] M. I. Shamos and G. Yuval. Lower bound from complex function theory. In *Proc. 17th IEEE Symp. on Foundations of Computer Science*, pages 268–273, October 1976.

[SY82] J.M. Steele and A.C. Yao. Lower bounds for algebraic decison trees. *Journal of Algorithms*, 3:1–8, 1982.

[Yao75] A.C. Yao. On the complexity of comparison problems using linear functions. In *Proc. 16 IEEE Symp. on Foundations of Computer Science*, pages 85–89, 1975.

[Yao89] A. C. Yao. Lower bounds for algebraic computaion trees with integer inputs. In *Proc. 30 IEEE Symp. on Foundations of Computer Science*, pages 308–313, Aug 1989.

[YR80] A. C. Yao and R. L. Rivest. On the polyhedral decision problem. *SIAM J. Comput.*, pages 343–347, 1980.

[DL78] D. Dobkin and R. J. Lipton. A lower bound of $\frac{1}{2}n^2$ on linear search programs for te knapsack problem. *J. Comp. and Syst. Sci.*, 16:413–417, 1978.

[DL80] D. Dobkin and R. J. Lipton. On the complexity of varying sets primatives. *J. Comp. and Syst. Sci.*, 18:86–91, 1980.

[DO85] E. Dittert and M. O'Donnell. Lower bounds for sorting with realistic instruction sets. *IEEE Transactions on Computers*, 34(4):311–317, April 1985. See also, Correction to: Lower bounds for sorting with realistic instruction sets. *IEEE Transactions on Computers*, 35(10):932, October 1986.

[Dob76] D. Dobkin. A non-linear lower bound on linear search tree programs for solving knapsack problems. *J. Comp. and Syst. Sci.*, 13:69–73, 1976.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, Berlin, 1988.

[IMR83] O. H. Ibarra, S. Moran, and L. E. Rosier. On the control power of integer division. *Theoretical Computer Science*, 24:35–52, 1983.

[Ja'81] J. Ja'Ja'. Computation of algebraic function with root extraction. In *Proc. 22nd IEEE Symp. on Foundations of Computer Science*, pages 95–100, October 1981.

[Jia79] Hong Jianwei. On lower bounds of time complexity of some algorithms. *Scientia Sinica*, pages 890–900, 1979.

[JMW89] B. Just, F. Meyer auf der Heide, and A. Wigderson. On computations with integer division. *RAIRO Informatique Theoretique et Applications*, 23(1):101–111, 1989. Also in, *Proc. 5th STACS, Lect. Notes in Comp. Sci.* No. 294 , pp. 29-37, 1988.

[KMR84] R. Kanna, G. Miller, and L. Rudolph. Sublinear parallel algorithm for computing the greatest common divisor of two integers. In 25th *Annual Symposium*

are relatively prime. An upper bound for this problem can be obtained from the parallel algorithms of [CG90, KMR84]. Analyzing those algorithms, with respect to the number of arithmetic operations, reveals an upper bound of $O(n/\log n)$ for computing the gcd.

Notice that the *gcd* can be written as an Integer Linear Program. Therefore, one of the consequences of our results is that there is no algorithm for the Integer Linear Programming problem, using operations only from the set $\{+, -, *, /, \lfloor\cdot\rfloor\}$, whose running time depends only on the number of variables and the number of constraints, and *not* on the size of the coefficients.

In Chapter 5 we prove lower bounds for approximating the s^{th} root of a real number. We show that approximating the s^{th} root, such that the average error is less than ϵ, requires at least $\Omega(\sqrt{\log\log\frac{1}{\epsilon}})$ operations. On the other hand, in Chapter 6, we prove an $O(\sqrt{\log\log\frac{1}{\epsilon}})$ upper bound, when the computation receives both the number to approximate and the accuracy parameter ϵ. This procedure approximates within ϵ in the worst case (i.e. for each input the approximated value is within ϵ of the correct value). Bashouty [Bas90b] has shown an $\Omega(\log\log\frac{1}{\epsilon})$ lower bound for worst case approximation. It still remains as an open problem the complexity of the average approximation for the s^{th} root.

We know that if the intermediate results in the computation can be exponential in the size of the input then any PSPACE computation can be done in a polynomial number of operations (see [BMS81]). On the other hand, if the intermediate results are restricted to be polynomial in the size of the inputs, then any computation with a polynomial number of operations is in polynomial time. One very interesting direction of research would be to show tradeoffs between the size of the intermediate results and the number of operations required.

$x_0 = 2^{n/2}$ and the constant $2^l = 2^{2^{(l+1)^2}}$, we establish the following theorem.

Theorem 6.6 *For infinitely many n, there is a decision tree with $OP = \{+, -, *, /, \lfloor\cdot\rfloor\}$ and constants $\{0, 1\}$ that decides if an integer x, $2^{n-1} \leq x < 2^n$, is a perfect square, and has depth $O(\sqrt{\log n})$. Furthermore, any decision tree that decides perfect squares (for any n) has depth $\Omega(\sqrt{\log n})$.*

```
input:          a
initialization: $b_1 = 2^2$, $a_1 = (2a)^2$, $c_1 = (2a)^4$, $d_1 = 2^2$, and $i = 1$.
computation:
                WHILE $b_i = d_i$ DO
                      $i = i + 1$;
                      $a_i = a_{i-1}^4$;      $b_i = a_i \bmod (2a - b_{i-1})$
                      $c_i = c_{i-1}^{16}$;     $d_i = c_i \bmod (a_i - d_{i-1})$
                END
output $d_i$    /* $d_i = 2^{2^{i^2}}$ */
```

Figure 6-3: Computing powers of two

terminating when they are not equal. It remains to show how to construct d_i.

In addition to the computation of a_i and b_i, we compute c_i and d_i, where: $c_1 = (2a)^4$, $d_1 = 2^2$, $c_i = c_{i-1}^{16}$ and $d_i = c_i \bmod (a_i - d_{i-1})$. It is easy to see that $c_i = (2a)^{2^{4i-2}}$. Notice that $c_i = a_i^{2^{2i-1}}$. Suppose that $b_{i-1} = d_{i-1} = 2^{2^{(i-1)^2}}$, and consider the i^{th} iteration:

$$d_i \equiv d_{i-1}^{2^{2i-1}} \bmod (a_i - d_{i-1}).$$

This implies that if $d_{i-1}^{2^{2i-1}} < a_i - d_{i-1}$ then $d_i = 2^{2^{i^2}}$. Clearly, if $d_{i-1}^{2^{2i-1}} < 2a - b_{i-1}$ and $b_{i-1} = d_{i-1}$, then $d_{i-1}^{2^{2i-1}} < a_i - d_{i-1}$. We conclude that for all i such that $2^{2^{i^2}} < 2a - b_{i-1}$, $b_i = d_i = 2^{2^{i^2}}$. Consider the *least* i such that $2^{2^{i^2}} > 2a - b_{i-1}$. Clearly, for this i, $b_i \neq 2^{2^{i^2}}$. However, we claim that d_i is still equal $2^{2^{i^2}}$. This is true since $2^{2^{(i-1)^2}} < 2a - b_{i-1}$, implying that

$$2^{2^{i^2}} = (2^{2^{(i-1)^2}})^{2^{2i-1}} < (2a - b_{i-1})^{2^{2i-1}} < (2a)^{2^{2i-1}} - b_{i-1}^{2^{2i-1}} = a_i - d_{i-1}.$$

Thus, the *first* time $b_i \neq d_i$, we stop the computation. Let t be the last index such that $b_t = d_t$, this guarantees that $b_t = 2^{2^{t^2}}$, and $a < 2^{2^{(t+1)^2}}$ (Figure 6-3 gives a full description of our algorithm for computing powers.)

such that if $\sqrt{x} \leq x_0 \leq \frac{3}{2}\sqrt{x}$ *and* $2^l > x$, *then*

$$|T(x, x_0, 2^l) - \sqrt{x}| \leq 1.$$

Proof: Let $n = l$, $d_i = \deg(H_i)$ and $M_i = \textit{max-coef}(H_i)$. We show how to compute $2^m \geq 2M_t 2^{2nd_t}$ in $O(t)$ steps. Once we show this, the rest of the algorithm and the proof is identical to the approximation case. Recall that $M_t \leq (M_0 + 1)^{2^{2t}}$. Since $x < 2^l$, then $M_0 \leq 2^l$, which implies that $M_t \leq (2^{l+1})^{2^{2t}} = M$. Since $d_t = 2^t$ and $2^n \leq 2^l$, we can compute $2^m = 2(2^l)^{2^{2t+1}} \geq 2M2^{2d_t n}$ in $O(t)$ steps. For $t = O(\sqrt{\log \log x})$, the above computation time is $O(\sqrt{\log \log x})$. Therefore the total running time of the entire procedure is $O(\sqrt{\log \log x})$. □

Later we show how to use the above lemma to prove that for infinitely many n, for any n bit integer, we can compute a "good" initial point in $O(\sqrt{\log n})$ steps.

6.3 Computing powers of two

Suppose that we want to compute $2^{2^{i^2}}$, for some i. Intuitively, it seems that the fastest way to do it is by successive squaring. This gives an $O(i^2)$ step procedure for computing this number. Surprisingly, using the floor operation, and given a large enough number, we can compute this power in $O(i)$ steps.

The main idea is based on the following observation. Let a, b and k be positive integers, and $a > b$. Then,

$$a^k \bmod (a-b) \equiv ba^{k-1} \bmod (a-b) \equiv b^k \bmod (a-b).$$

If $b^k < a - b$ then the value of the above computation is simply b^k. That suggests that once we raised a to the k^{th} power, we can raise b to the k^{th} power in $O(1)$ operations. By already computed 2^m, we can compute $c_{H_t}^m = H_t(2^m)$ in $O(t)$ steps. To summarize, so far we have computed 2^m, 2^{md} and $c_{H_t}^m$ in $O(t)$ steps. Thus, we have computed all the constants necessary for Theorem 6.1, and therefore an evaluation of H_t can be performed in $O(1)$ operations.

The computation evaluates H_t k times; the input to the i^{th} evaluation of H_t, x_i, is the output of the $(i\text{-}1)^{th}$ computation, i.e. $x_i = H_t(x_{i-1})$. Each evaluation of H_t costs only $O(1)$, therefore the length of the program is $O(t+k)$, and it performs tk Newton iterations. This implies that it can perform $O(\log\log\frac{1}{\epsilon})$ iterations in $O(\sqrt{\log\log\frac{1}{\epsilon}})$ steps (by choosing $t = k$).

However, we are still left with a slight implementation problem. Notice that x_i, for $i > 0$, may be non-integer, while the polynomial evaluation procedure is valid *only* if x_i is an integer. (Another condition is that $x_i < 2^n$, but this is always true.) To fix this problem, suppose that in each iteration instead of computing x_i, we compute the smallest integer greater than x_i. Formally, define the modified iteration as follows.

$$\hat{x}_{i+1} = \left\lceil \frac{F_t(\hat{x}_i)}{G_t(\hat{x}_i)} \right\rceil$$

We claim that this does not change the convergence rate by more than a constant factor. Let $\hat{x}_i = (1+\hat{\delta}_i)\sqrt{a}$. It can be verified that,

$$|\hat{\delta}_{i+1}| \leq 2(\frac{\hat{\delta}_i}{2})^{2^t} + \frac{1}{\sqrt{a}}.$$

As long as $\hat{\delta}_i > \frac{2}{\sqrt{a}}$, this implies that $2(\frac{\hat{\delta}_{i-1}}{2})^{2^t} \geq \frac{1}{\sqrt{a}}$, and therefore,

$$\hat{\delta}_i \leq 4(\frac{\hat{\delta}_{i-1}}{2})^{2^t} \leq \hat{\delta}_0^{2^{it}}$$

On the other hand, if $0 \leq \hat{\delta}_i \leq \frac{2}{\sqrt{a}}$, then $\sqrt{a} \leq \hat{x}_i \leq \sqrt{a}+2$. This implies that $|\sqrt{a} - (\hat{x}_i - 1)| \leq 1$. Therefore, we can set $\alpha = \hat{x}_i - 1$, and the approximation to $\sqrt{b}$

```
input:          b,$\epsilon$
computation:
                $a = \left\lfloor \frac{4b}{\epsilon^2} \right\rfloor$
                $\alpha = R_t(a)$/* invoke the polynomial evaluation procedure */
                $\beta = \frac{\alpha\epsilon}{2}$
output $\beta$.
```

Figure 6-1: Approximating the square root non-uniformly

$Q_1(y) = 2x_0$. In general, x_i is obtained by evaluating a rational function

$$R_i(y) = \frac{P_i(y)}{Q_i(y)} = \frac{y + (\frac{P_{i-1}(y)}{Q_{i-1}(y)})^2}{2\frac{P_{i-1}(y)}{Q_{i-1}(y)}} = \frac{yQ^2_{i-1}(y) + P^2_{i-1}(y)}{2P_{i-1}(y)Q_{i-1}(y)}$$

at the point $y = a$. This implies that $P_i(y) = yQ^2_{i-1}(y) + P^2_{i-1}(y)$ and $Q_i(y) = 2P_{i-1}(y)$ $Q_{i-1}(y)$, for $i > 1$. (Note that the coefficients of P_i and Q_i are integers.) We are interested in computing $\alpha = R_t(a) = P_t(a)/Q_t(a)$. Note that given x_0 and t the polynomials P_t and Q_t are defined. Furthermore, x_0 depends only on ϵ, and not on a. Since, the coefficients of P_i and Q_i are integers, by Theorem 6.1, we can compute $P_t(a)$ and $Q_t(a)$ in constant time, for any n-bit integer a. Hence, we can compute $\alpha = R_t(a)$ in constant time.

Let $\beta = \alpha\epsilon/2$. Since α can be computed in $O(1)$ steps, β can be computed in $O(1)$ steps. As we discussed before, $|\sqrt{b} - \beta| \leq \epsilon$, therefore we have shown the following theorem.

Theorem 6.2 *There exists a constant C such that, for any $0 < \epsilon < 1$ there is a straight line program L_ϵ of length C, such that for any $b \in [1, 2]$,*

$$|\sqrt{b} - L_\epsilon(b)| \leq \epsilon$$

of $\gamma(a)c_p^m$. More formally, compute

$$\beta = \left\lfloor \gamma(a)c_p^m 2^{-md} \right\rfloor - \left\lfloor \gamma(a)c_p^m 2^{-md-m} \right\rfloor 2^m.$$

We claim that $P(a) = \beta$. Consider the product $\gamma(a)c_p^m$, then

$$\begin{aligned}
\gamma(a)c_p^m &= \sum_{i=0}^{d}\sum_{j=0}^{d} p_i a^j 2^{m((i-j)+d)} \\
&= \sum_{k=-d}^{d} (\sum_{i=\max\{0,k\}}^{\min\{d,d+k\}} p_i a^{i-k}) 2^{m(k+d)} \\
&= \sum_{k=-d}^{d} g_k 2^{m(k+d)}
\end{aligned}$$

Note that $P(a) = \sum_{i=0}^{d} p_i a^i = g_0$. For m, such that $2^m > \max_i\{g_i\}$, the m-bit integer obtained by considering bit positions md to $m(d+1)-1$ of $\gamma(a)c_p^m$ is $g_0 = \beta$. This concludes the description of the polynomial evaluation algorithm. Below we bound the value of m that meets the above requirements.

Let, *max-coef*$(P) = M$. Since $a < 2^n$, then the value of a^i is bounded by 2^{ni}. Thus, each g_k is bounded by $dM2^{nd}$. In step 1 we required that $m \geq n(d+1)+1$, therefore it is sufficient to chose $m \geq 2nd + \lceil \log M \rceil + 1$. (Note that this implies that the size of the constants used are polynomial in d, n and $\log M$.)

Recall that a straight line program is a computation tree without any comparison nodes. Thus, we have shown the following theorem.

Theorem 6.1 *There exists a fixed constant C such that, for any polynomial P with integer coefficients, there is a straight line program L of length C, that computes $P(x)$ for all n-bit positive integers. If* $\deg(P) = d$ *and max-coef$(P) = M$, then the constants used in L are 2^m, 2^{md}, and c_p^m, for any $m \geq 2nd + \lceil \log M \rceil + 1$.*

of the polynomial. In section 6.2 we apply the techniques developed for evaluation of polynomials to the problem of approximation of square roots. For the non-uniform case, we show a straight line program that approximates the square root in $O(1)$ steps. For the uniform case, we show a computation tree T that, on input $\epsilon > 0$ and $x \in [1,2]$ computes $T(\epsilon, x)$ such that $|\sqrt{x} - T(\epsilon, x)| \leq \epsilon$, and the has length $O(\sqrt{\log\log \frac{1}{\epsilon}})$. In contrast, it is known that using only rational operation, $\Theta(\log\log \frac{1}{\epsilon})$ steps are both necessary and sufficient (see [Bra86]).

Note that when we are considering approximation of square roots, we are considering two classes of algorithms. The first class is algorithms that receive only the input, x, and although they have a guarantee that the output is at most ϵ from the square root, they do not use the value of ϵ in the computation. An example of such an algorithm is the Newton iteration method; the lower bound that we show is for this class of algorithms. The second class, is an algorithm that has two inputs, the point x, and the desired approximation ϵ. Unlike the previous type, here the algorithm uses the value of ϵ during the computation. The upper bound belongs to the second class of algorithms.

Section 6.3 shows that the floor function can be used to accelerate exponentiation. Suppose we would like to generate the number $2^{2^{i^2}}$. Using repeated squaring this can be done in $O(i^2)$ steps. We show that this number can be generated in $O(i)$ steps, given any integer $x > 2^{2^{i^2}}$ (and using the floor function). Using this procedure we show that part of the lower bounds proved in Chapter 3 are tight.

6.1 Computing polynomials

We consider the complexity of evaluating a polynomial, whose coefficients are integers, on an n-bit integer input. We show that this problem can be solved in constant time, using arbitrary constants and the floor function.

starts at v_1, and ends at v_{i+2}. We choose $[a_{i+1}, a_{i+1} + l_{i+1}]$ as a subinterval of $[a_i, a_i + l_i]$. Therefore, by the induction hypothesis we have that Properties 1-3 are satisfied by the interval $[a_{i+1}, a_{i+1} + l_{i+1}]$ and the prefix of $\mathcal{P}$ that starts at v_1 and ends at v_{i+1}. In order to complete the proof of the lemma we need to show that (a) there exists an outgoing edge of v_{i+1} such that for any input $x \in [a_{i+1}, a_{i+1} + l_{i+1}]$ the computation follows this edge, and (b) Properties 2-3 are satisfied also for the vertex v_{i+1} and the interval $[a_{i+1}, a_{i+1} + l_{i+1}]$.

By the definition of the tree T, the vertex v_{i+1} is either a comparison vertex or a computation vertex. If it is a comparison vertex, then a comparison $g(x) \leq h(x)$ is performed. If it is a computation vertex, then either $f_{v_{i+1}} = r_{i+1}(x) = g(x) \circ h(x)$ for $\circ \in \{+, -, *, /\}$, or $f_{v_{i+1}} = \lfloor g(x) \rfloor$ is evaluated. Here, $g(x), h(x) \in \{0, 1\} \cup \{f_{v_j} | v_j$ is a computation vertex, $j \leq i\}$.

By the induction hypothesis, $g(x)$ and $h(x)$ are rational functions of degree less than D_i and *max-coef*(g), *max-coef*$(h) \leq M_i$.

The proof is based on a case by case analysis. In each case, we define the next vertex v_{i+2} on the path $\mathcal{P}$, the interval $[a_{i+1}, a_{i+1} + l_{i+1}]$, and the rational function $r_{i+1}(x)$ (whenever v_{i+1} a computation vertex).

First, we resolve the case when v_{i+1} is a comparison vertex. The comparison is of the form $g(x) \leq h(x)$. By the induction hypothesis, both $h(x)$ and $g(x)$ can be represented as polynomials of degree less than D_i. Consider the rational function $g(x) - h(x)$, which is of degree less than $2D_i$. Use Lemma 5.2 and choose a subinterval $[a_{i+1}, a_{i+1} + \frac{l_i}{14D_i}]$ where $g(x) - h(x)$ does not have any poles or zeros. Set $l_{i+1} = \frac{l_i}{14D_i}$. It is now easy to check that properties 1-3 are satisfied in this case.

Next, consider the case when v_{i+1} is a computation vertex. The following possibilities may arise.

Suppose that $\underline{\circ \in \{+, -, *, /\}}$. Let $r_{i+1}(x) = g(x) \circ h(x)$, $a_{i+1} = a_i$, and $l_{i+1} = l_i$. By Lemma 2.1 the degree of $r_{i+1}(x)$ is less than $2D_i$, and its maximum coefficient is *max-coef(r)* $\leq M$, *there exists a subinterval* $[\alpha, \alpha + \frac{l}{448(sd+1)^4}]$ *such that for any* v *in the subinterval,* $|v^{1/s} - r(v)| \geq (2(2M)^2)^{-1}(\frac{l}{32s})^{(sd+1)}$.

Proof: Let $x^s = v$, such that $\sqrt[s]{a} \leq x \leq \sqrt[s]{a} + \frac{l}{4s} \leq \sqrt[s]{a+l}$ Consider the rational function $E(x) = x - r(x^s)$. Clearly, the degree of $E(x)$ is bounded by sd+1 and *max-coef*$(E) \leq 2M$.

By Corollary 5.8, there exists a subinterval $[\beta, \beta + \lambda] \subset [\sqrt[s]{a}, \sqrt[s]{a} + \frac{l}{4s}]$, such that $\lambda = \frac{l}{224s(sd+1)^4}$, and for any $x \in [\beta, \beta + \lambda]$, $|E(x)| \geq (2(2M)^2)^{-1}(\frac{l}{32s})^{(sd+1)}$.

Let $\alpha = \beta^s$ and the subinterval be $[\alpha, \alpha + s\lambda/2]$. For any $v \in [\alpha, \alpha + s\lambda/2]$, then $x \in [\beta, \beta + \lambda]$. This implies that on the subinterval $[\alpha, \alpha + s\lambda/2]$, then difference between $v^{1/s}$ and $r(v)$ is at least $(2(2M)^2)^{-1}(\frac{l}{32s})^{(sd+1)}$. □

5.5 Lower bound for approximation

In this section we use the technique that we developed in the previous section to establish the lower bound on approximating $\sqrt[s]{x}$. We show an $\Omega(\sqrt{\log\log\frac{1}{\epsilon}})$ lower bound on the depth of any computation tree T, with $OP = \{+, -, *, /, \lfloor\cdot\rfloor\}$ and constants $\{0, 1\}$, that ϵ-approximates $\sqrt[s]{x}$, where s is any fixed constant, for $x \in [1, 2]$, in norm L_1.

Theorem 5.11 *For any integer* $s \geq 2$, *if a computation tree* T, *with operations* $OP = \{+, -, *, /, \lfloor\cdot\rfloor\}$ *and constants* $\{0, 1\}$, *approximates* $\sqrt[s]{x}$, *for a fixed* s *and* $x \in [1, 2]$, *such that*

$$error_1(T(x), \sqrt[s]{x}) \leq \epsilon$$

then T *has depth* $h = \Omega(\sqrt{\log\log\frac{1}{\epsilon}})$.

Proof: The bulk of the proof involves constructing (i) a path $\mathcal{P}$ from the root of T to one of its leaves, and (ii) a subinterval $[a, a + l]$ of $[1, 2]$, with the following properties:

1. On input $\alpha \in [a, a + l]$ to T, the computation follows the path $\mathcal{P}$; and

Note that $q(z)$ and $p(x)$ have the same degree and,

$$|b|(\frac{l}{2})^d \max_{z\in[-1,1]}\{|q(z)|\} = \max_{x\in[a,a+l]}\{|p(x)|\}.$$

By Lemma 5.5, there exists $z_0 \in [-1,1]$ such that $|q(z_0)| \geq 2^{-d+1}$. Therefore, $K \geq 2^{-d+1}|b|\ (\frac{l}{2})^d$. (Recall that $|b| \geq 1$.) Hence $K/2 \geq (\frac{l}{4})^d$, which completes the proof. □

The following lemma combines the above lemma, which enables us to give lower bounds for the value of a polynomial, with the first lemma, that observed how to upper bound a polynomial, to establish a technique to upper bound a rational expression.

Lemma 5.7 *Given an interval $[a, a+l]$, where $1 \leq a \leq 2$ and $0 < l \leq 2-a$, and a rational function $r(x)$ of degree d with* integer *coefficients and max-coef$(r) \leq M$; there exists a subinterval $[\alpha, \alpha + \frac{l}{56d^4}]$, such that (i) $r(x)$ is monotone in this subinterval, and (ii) $|r(x)| \leq 2M(\frac{8}{l})^d$ for all $x \in [\alpha, \alpha + \frac{l}{56d^4}]$.*

Proof: Let $r(x) = \frac{p(x)}{q(x)}$ where $p(x)$ and $q(x)$ are polynomials of degree at most d with integer coefficients. By Lemma 5.1, the maximum value of $|p(x)|$ for $x \in [1,2]$ is bounded from above by $M2^{d+1}$. By Lemma 5.2, there is a subinterval $[\beta, \beta + \frac{l}{7d}]$ in which $r(x)$ is monotone. By Lemma 5.6, there is a subinterval of $[\beta, \beta + \frac{l}{7d}]$, denoted by $[\alpha, \alpha + \frac{l}{56d^4}]$, such that $|q(x)| \geq (\frac{l}{4})^d$ for all $x \in [\alpha, \alpha + \frac{l}{56d^4}]$. Therefore, $|r(x)| \leq 2M(\frac{8}{l})^d$ for all $x \in [\alpha, \alpha + \frac{l}{56d^4}]$. □

Let $r_1(x)$ be a rational expression and $r_2(x) = \frac{1}{r_1(x)}$. Since both the degree and the maximum coefficient of r_1 and r_2 are identical, a lower bound for the value of r_1 would imply an upper bound for the value of r_2. The following corollary formalizes this observation.

Corollary 5.8 *Given an interval $[a, a+l]$, where $1 \leq a \leq 2$ and $0 < l \leq 2-a$, and a rational function $r(x)$ of degree d with* integer *coefficients and max-coef$(r) \leq M$; there*

We need to use a slight modification of Markoff inequality. The segments that we are interested in are not necessarily $[-1,+1]$. Therefore the following formulation would be more convenient.

Lemma 5.4 *Let $p(x)$ be a polynomial of degree d, and $p'(x)$ the derivative of $p(x)$. Then*

$$\max_{x\in[a,a+l]}\{|p'(x)|\} \le \frac{2d^2}{l} \max_{x\in[a,a+l]}\{|p(x)|\}.$$

Proof: Consider the substitution $x = z\frac{l}{2} + \frac{2a+l}{2}$. If $z \in [-1,1]$ then $x \in [a, a+l]$. Let $q(z) = p(z\frac{l}{2} + \frac{2a+l}{2})$. By Lemma 5.3

$$\max_{z\in[-1,1]}\{|q'(z)|\} \le d^2 \max_{z\in[-1,1]}\{|q(z)|\}.$$

Since $\frac{dq}{dz}\frac{dz}{dx} = \frac{dp}{dx}$, and $\frac{dz}{dx} = \frac{2}{l}$, the lemma follows. □

The importance of this lemma is that using it we can claim, that if over the interval $[a, a+l]$ the maximum value of a polynomial is $p(x')$, then there is a neighborhood of x', in which the value of the polynomial is $\Omega(p(x'))$ and the size of the neighborhood is $\Omega(l/d^2)$.

The family of polynomials, known as Chebyshev polynomials, have many application in Numerical Analysis. We use here only one aspect of the Chebychev polynomials, and that is their ability to give a lower bound for the value of a polynomial.

Lemma 5.5 (Chebyshev [Riv69, page 31]) *Let $p(x)$ be a polynomial of degree $d \ge 1$ defined over $[-1,1]$. If the leading coefficient of $p(x)$ is one, then there exists $x_0 \in [-1,1]$ such that $|p(x_0)| \ge \frac{1}{2^{d-1}}$.*

The importance of the above lemma is to enable us to establish a lower bound on the value of a polynomial. Once we are able to show a lower bound and upper bounds proving a lower bound for rational expression, over $[a, a+l]$.

The last part of the proof involves proving lower bounds for rational functions. We show that for a rational expression $r(x)$, defined over $[a, a+l]$, there is a subinterval $[\alpha, \alpha+\lambda]$, such that for any input $x \in [\alpha, \alpha+\lambda]$, the difference between $r(x)$ and $\sqrt[s]{x}$ is at least δ. This implies that

$$\|r(x) - \sqrt[s]{x}\|_1 \geq \delta\lambda.$$

5.4 Lower bound technique

We would like to establish a technique to bound the value of a rational function over an interval. In general, a rational function may be unbounded; therefore we are seeking a "large" subinterval in which the rational function is bounded. The techniques that we develop show how to both upper bound and lower bound the value of a polynomial. The first lemma shows a very trivial upper bound on the value of a polynomial defined over $[1, 2]$. Much of the remainder of the section is devoted to showing that given a polynomial, there is a subinterval in which we can lower bound the value of the polynomial.

Lemma 5.1 *Let $p(x)$ be a polynomial of degree d and max-coef$(p) \leq M$. Then, for all $x \in [1, 2]$, $|p(x)| < M2^{d+1}$.*

Proof: Let $p(x) = \sum_{i=0}^{d} a_i x^i$. Then clearly,

$$|p(x)| = |\sum_{i=0}^{d} a_i x^i| \leq \sum_{i=0}^{d} |a_i| x^i \leq \sum_{i=0}^{d} M2^i < M2^{d+1}$$

□

It would be convenient that the rational functions that we are dealing with would not have any poles (i.e. inputs for which they go to infinity) or any roots. The following

Recently, Bashouty [Bas90b] has shown that approximating the square root of two, requires $\Omega(\log\log\frac{1}{\epsilon})$ operations, when the only input is x, and ϵ is not an input. This implies a lower bound in norm L_∞. It is still an open problem what is the bound for norm L_1, where the only input is the number to be approximated, and not ϵ.

5.3 Overview

In this section we give a general overview of the proof that we show in the next two sections. At a very high level, we establish the proof along the following lines. First, we show that there is a "large" subinterval, such that all the inputs from this subinterval follow the same path. Second, the output that the computation tree computes for this subinterval can be expressed as a rational function. Third, any rational function, of a given degree and coefficient size, would have to be "far" from the s^{th} root function, on a "substantial" part of the subinterval. Clearly we would have a tradeoff between how "far" the functions are, and how "substantial" is the part of the subinterval on which they are "far". Combining those two parameters translates to a lower bound on the approximation in norm L_1.

Now we elaborate slightly more on each component of the proof. We start with the main tool that enables us to show how to handle the floor function. Consider the expression $\lfloor r(x) \rfloor$, where r is a rational expression, defined over $[a, a+l]$. We show that there is a sub-interval, $[a', a'+l']$, such that the value of $\lfloor r(x) \rfloor$ is constant in it and the ratio between l and l' is bounded as a function of *deg*(r) and *max-coef*(r). The smaller the ratio between l' and l, the better lower bound we can prove at the end. The proof of this claim quite involved, and uses theorems from Approximation Theory, such as Chebyshev polynomials and the Markoff inequality.

Given a computation tree T that receives an input $x \in [1,2]$, we show that there exists an interval, $[a, a+l] \subset [1,2]$, such that any input $x \in [a, a+l]$ follows the same example, that computing the average distance of n points in the plane requires exactly $\binom{n}{2}$ operations. (For average distance, [SY76] showed that it can be approximated with substantially less than $\binom{n}{2}$ operations.)

This work pursues the second direction. Rather than computing a function exactly, we are interested in approximating the function's value. The set of basic operations does not change and remains the set of rational operations and the floor function, i.e. $\{+,-,*,/,\lfloor\cdot\rfloor\}$.

This chapter is organized as follows. In the following section we define formally the meaning of an approximation. In section 5.3 we give a general overview of the lower bound technique that we develop. In section 5.4 we develop the lower bound technique, that we later use in Section 5.5 to prove the lower bound for approximating the s^{th} root of a real number.

5.2 Definitions

The main aim of this section is to define formally the notion of approximation. We start by a very brief background, and define inner product of real functions, and a norm of a real function. The *inner product* of two real functions f and g, over a region Λ, is defined as follows,

$$< f,g >= \int_\Lambda f(x)g(x)dx$$

Given the definition of the inner product, We define the *norm* L_k, of a function f, over a region Λ, to be

$$\|f\|_k = \sqrt[k]{\int_\Lambda |f(x)|^k dx}$$

tree are input vertices of the inputs a and b. If (u, v) is the generator of (a, b), then, by Lemma 4.6, a and b are polynomials in u and v of maximum coefficient and degree less than $2^{n^{1/5}}$ and $n^{1/5}$, respectively. This implies that the number of monomials in each of these polynomials is at most $n^{2/5}$ and that the value of each monomial is at most $2^{n^{1/5}}(2^{2n^{2/5}})^{2n^{1/5}}$. Therefore, a, $b \leq n^{2/5}2^{n^{1/5}}2^{2n^{2/5}2n^{1/5}} = n^{2/5}2^{4n^{4/5}} < 2^n$, for large enough n.

Lemma 4.7 asserts that some pairs in the set $(u, v) \in S(0, \alpha_1, \alpha_2, \alpha_3) \cap \{(u, v) : 1 \leq u, v < 2^{2t(t+1)}\}$ are relatively prime, and some are not. The Correspondence Property, Lemma 4.1, guarantees that the gcd of the inputs is the same as the gcd of the generators. Since the leaf ν is labeled by a constant, the inputs that reach this leaf are either all relatively prime, or all not relatively prime. Since all the pairs reach the same leaf, we reach a contradiction. □

(u, v) is the generator of (x, y). By Lemma 4.4, the degree of $\frac{F_\nu(u,v)}{G_\nu(u,v)} \circ \frac{F_\mu(u,v)}{G_\mu(u,v)}$ is less than $2D_i \leq 2^{2^{4(i+1)}}$, and its maximum coefficient is bounded by $2M_i^2(D_i + 1)^2 \leq 2^{2^{2^{4(i+1)}}}$.

Suppose v_{i+1} is $\lfloor \nu \rfloor$, where ν is a previous computation vertex. By the induction hypothesis, $val(\nu) = \frac{F_\nu(u,v)}{G_\nu(u,v)}$, where (u, v) is the generator of (x, y). By Lemma 4.5, there is a rational expression $Q(x, y)$, and a subset of the inputs $\mathcal{S}^{(i+1)}$, such that $val(\nu) = Q(u, v)$, where (u, v) is the generator of (x, y). The degree of Q is at most $D_{i+1} \leq 4D_i^4 \leq 2^{2^{4(i+1)}}$, and the maximum coefficient of Q is at most $M_{i+1} \leq 2(4D_iM_i)^{2D^3} \leq 2^{2^{2^{4(i+1)}}}$.

We are not done yet with this case. The substitution of u_i by $\lambda_{i+1}(u_{i+1})^{\delta_{i+1}} + u_{i+2}$, affects not only the rational expression Q, which is stated in Lemma 4.5. It also changes *all* the polynomials in Σ_i. Since $\delta_{i+1} \leq D_i + 1$, the degree of the rational expressions in Σ_i is at most $D_{i+1} \leq D_i(D_i + 1)$, and the maximum coefficient is at most $M_{i+1} \leq \lambda_{i+1}^{D_i} 2^{D_i} M_i$. Note that the degree and maximum coefficients of previous vertices change, but they are bounded by the current D_i and M_i. □

4.4 GCD Lower bound

In this section we show how to apply the proof technique that we developed in the previous section to derive a lower bound for computing the greatest common divisor of two integers.

We start by showing that in any set of inputs $S(\cdot)$, there is one input that is relatively prime, and another one that is not relatively prime.

Lemma 4.7 *Let $\alpha_1, \alpha_2, \alpha_3$, and t be positive integers such that $\alpha_1 < t$, and $\alpha_2, \alpha_3 < 2^t$. Then the set $S(0, \alpha_1, \alpha_2, \alpha_3) \cap \{(u, v) : 0 < u, v < 2^{2t(t+1)}\}$ contains two pairs (a_0, a_1) and (b_0, b_1), such that* $\gcd(a_0, a_1) \neq 1$ *and* $\gcd(b_0, b_1) = 1$.

Proof: Let e be the least positive integer exponent such that $(1 + \alpha_3)^e > \alpha_2$. Define

Lemma 4.6 *Let T be a computation tree of depth h, with two inputs, the operations $\{+, -, *, /, \lfloor\cdot\rfloor\}$ and constants $\{0, 1\}$. Then, there is a path $\mathcal{P}$ from the root of T to a leaf, and a subset $\mathcal{S}$ of inputs, with the the following properties:*

1. *$\mathcal{S} = S(r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda) \cap \{(a, b) : 0 < a, b < 2^n\}$, for some positive integers $r, \alpha_1, \alpha_2, \alpha_3, \delta_1, \delta_2, \ldots, \delta_r$, $\lambda_1, \lambda_2, \ldots, \lambda_r$, where $\Delta = (\delta_1, \delta_2, \ldots, \delta_r)$, and $\Lambda = (\lambda_1, \lambda_2, \ldots, \lambda_r)$;*
2. *For each input $(a, b) \in \mathcal{S}$. the computation follows the path $\mathcal{P}$;*
3. *For each computation or input vertex ν on the path $\mathcal{P}$, there is a pair of bivariate polynomials $(F_\nu(x, y), G_\nu(x, y))$ with integer coefficients, such that for each input $(a, b) \in \mathcal{S}$, $G_\nu(u, v) \neq 0$, and $val(\nu) = \frac{F_\nu(u,v)}{G_\nu(u,v)}$, where (u, v) is the $<r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda>$-generator of (a, b); i.e., the value computed at ν on input $(a, b) \in \mathcal{S}$, is the value of the rational expression $\frac{F_\nu(x,y)}{G_\nu(x,y)}$ at (u, v); and*
4. *Let $\Sigma = \{F_\nu(x, y), G_\nu(x, y) \mid \nu \in \mathcal{P}\}$. Define D and M to be the degree and the maximum coefficient of Σ, respectively. Then, $r \leq h$, $\max\{\alpha_1, D\} < 2^{2^{4h}}$, and $\max\{\alpha_2, \alpha_3, M\} < 2^{2^{2^{4h}}}$.*

At first sight it might be surprising that the above lemma does not claim explicitly any bound on the values in Δ and Λ. The reason is that we can add the input as two input vertices in the tree T, and therefore, we express the input as a rational expression of the "generators". This guarantees the connection of the values in Σ with Δ and Λ.

Proof: We denote the vertices on the path $\mathcal{P}$ by $v_1, v_2, \ldots, v_l$, in that order, where v_1 is the root of the tree T, v_i is a child of v_{i-1}, and v_l is a leaf of the tree T. The path $\mathcal{P}$ and the set $\mathcal{S}$ are defined inductively, starting with the path v_1, v_2, v_3 and the set $\mathcal{S}^{(2)} = S(0, 1, 0, 1)$ (which consists of all pairs (a, b), where $a > b > 0$). Following that proof, suppose that (a) we have selected a prefix of $\mathcal{P}$, which starts at v_1, and ends at a vertex v_{i+1}, and (b) constructed the set $\mathcal{S}^{(i)} = S(r^{(i)}, \alpha_1^{(i)}, \alpha_2^{(i)}, \alpha_3^{(i)}, \Delta^{(i)}, \Lambda^{(i)})$ with the following properties:

and,

$$\begin{aligned}\alpha_2' = \max\{\hat{\alpha}_2, \pi_2(\tilde{R}), \pi_2(L^d P_2 - \tilde{R})\} &\leq M^{D+1} + ((D+2)M)^{D+2} + 1 \\ &\leq 2((D+2)M)^{D+2}.\end{aligned}$$

This guarantees that for any $(u,v) \in S(0, \alpha_1', \alpha_2', \pi\alpha_3)$, $0 < \tilde{R}(u,v)$ and $\tilde{R}(u,v) < L^d P_2(u,v)$. This implies that

$$0 < \frac{\tilde{R}(u,v)}{L^d P_2(u,v)} < 1.$$

We conclude that for $(x,y) \in S(0, \alpha_1', \alpha_2', \pi\alpha_3)$,

$$\lfloor P(x,y) \rfloor = \frac{A(x,y) - \gamma + \rho(R)L^d}{L^d} = Q(x,y)$$

Clearly, $deg(Q) = deg(A) \leq D$, and $max\text{-}coef(Q) \leq max\text{-}coef(A) + 2M^D \leq 2((2+D)M)^{D+1}$.

In Case 5 we show how to reduce the general case to Case 4. In order to summarize the effect of Case 4, observe that $\max\{\alpha_1', deg(Q)\} \leq (D+1)^2$, and $\max\{\alpha_2', \pi\alpha_3, max\text{-}coef(Q)\} \leq 2((D+2)M)^{D+2}$.

Case 5: The leading monomial of $P_2(x,y)$ is $Lx^{d_x}y^l$. Our goal is to reduce this case to Case 4 where no powers of y appear in the leading monomial. We introduce a new variable z and substitute x using it. Let $\delta = \max\{\alpha_1, deg(P_2) + 1\} \leq D + 1$ and $\lambda = \alpha_3$. we substitute x by $\lambda y^\delta + z$ and consider the polynomial $\hat{Q}(y,z) = P_2(\lambda y^\delta + z, y)$. Observe that the leading monomial in $\hat{Q}(y,z)$ is a constant times a power of y, i.e., no power of z appears in the leading monomial of $\hat{Q}(y,z)$. Thus, we have reduced this case to Case 4.

As in Case 3, let $A(1,1) = cL^d + \gamma$, where c is an integer and γ is a non-negative integer such that $0 \leq \gamma < L^d$.

$$\frac{P_1(u,v)}{P_2(u,v)} = \frac{\gamma P_2(u,v) + R(u,v)}{L^d P_2(u,v)} + \frac{A(u,v) - \gamma}{L^d}.$$

Let $\pi = L^d$. Then, for each $(u,v) \in S(0, \hat{\alpha}_1, \hat{\alpha}_2, \pi\alpha_3)$, $A(u,v) \equiv \gamma \pmod{L^d}$. Hence, for each such pair (u,v), the expression $\frac{A(u,v)-\gamma}{L^d}$ evaluates to an integer. Therefore,

$$\left\lfloor \frac{P_1(u,v)}{P_2(u,v)} \right\rfloor = \left\lfloor \frac{\gamma P_2(u,v) + R(u,v)}{L^d P_2(u,v)} \right\rfloor + \frac{A(u,v) - \gamma}{L^d}.$$

Our aim is to restrict the inputs such that the value of $\left\lfloor \frac{\gamma P_2(u,v)+R(u,v)}{L^d P_2(u,v)} \right\rfloor$ is fixed. We distinguish between two subcases, according to the value of γ.

Subcase 4.1: $\gamma > 0$. In this subcase we restrict the input, such that the value of $\left\lfloor \frac{\gamma P_2(u,v)+R(u,v)}{L^d P_2(u,v)} \right\rfloor$ is zero. Consider the polynomial $V(x,y) = L^{-d}(\gamma\ P_2(x,y) + R(x,y))$. Since $deg_x(R) < deg_x(P_2)$ the leading coefficient of V is $\frac{\gamma}{L^d}L$. Let $B(x,y) = P_2(x,y) - V(x,y)$. The leading coefficient of $B(x,y)$ is $(1 - \frac{\gamma}{L^d})L$. Since $0 < \frac{\gamma}{L^d} < 1$, the leading coefficient of $B(x,y)$ is positive, i.e. $sign(B) = +1$. Recall that $\pi = L^d$. Let,

$$\begin{aligned} \alpha_1' = \max\{\hat{\alpha}_1, \pi_1(B), \pi_1(\gamma P_2 + R)\} &\leq D(D+1)+1 \\ &\leq (D+1)^2, \end{aligned}$$

and,

$$\begin{aligned} \alpha_2' = \max\{\hat{\alpha}_2, \pi_2(B), \pi_2(\gamma P_2 + R)\} &\leq M^{D+1} + ((D+2)M)^{D+2} + 1 \\ &\leq 2\left((D+2)M\right)^{D+2}. \end{aligned}$$

3. $P_2(x,y)$ is a constant, i.e., $d_x = d_y = 0$.

4. $l = 0$, i.e. y does not appear in the leading monomial of $P_2(x,y)$.

5. $l > 0$, i.e. the general case.

Case 1: $P_1(x,y) = P_2(x,y)$, for $(x,y) \in S(0,\alpha_1,\alpha_2,\alpha_3)$ Let $Q(x,y) = 1/1$, $\pi = 1$, $\alpha_1' = \alpha_1$ and $\alpha_2' = \alpha_2$.

Case 2: $P_1(x,y) \prec P_2(x,y)$. Let $B(x,y) = P_2(x,y) - P_1(x,y)$. Since the leading coefficient of P_2 is positive and greater than the leading coefficient of P_1, then $sign(B) = +1$. Using Lemma 2.3 let $\alpha_1' = \max\{\hat{\alpha}_1, \pi_1(B)\}$ and $\alpha_2' = \max\{\hat{\alpha}_2, \pi_2(B)\}$. Lemma 2.3 guarantees that for each $(u,v) \in S(0,\alpha_1',\alpha_2',1)$, $B(u,v) > 0$. Observe that $B(u,v) > 0$ implies that $P_2(u,v) > P_1(u,v)$. Since $\alpha_1' \geq \pi_1(P_2)$ and $\alpha_2' \geq \pi_2(P_2)$, then $P_2(u,v) > 0$. Thus,

$$\frac{P_1(u,v)}{P_2(u,v)} < 1.$$

For each $(u,v) \in S(0,\alpha_1',\alpha_2',1)$ both $P_1(u,v) > 0$ and $P_2(u,v) > 0$. Therefore,

$$0 < \frac{P_1(u,v)}{P_2(u,v)} < 1.$$

Let $\pi = 1$. This implies that for $(x,y) \in S(0,\alpha_1',\alpha_2',1)$,

$$\left\lfloor \frac{P_1(u,v)}{P_2(u,v)} \right\rfloor = \frac{0}{1} = Q(x,y)$$

Clearly, $deg(Q) = 0$ and *max-coef*$(Q) = 1$. Since $deg(B) \leq D$, and *max-coef*$(B) \leq 2M$, then by Lemma 2.3, $\alpha_1' \leq D + 1$ and $\alpha_2' \leq 2M + 1$.

Case 3: $P_2(x,y)$ is the integer $L \geq 1$. Recall that all coefficients of $P_1(x,y)$ are integers, and $P_1(x,y) > 0$, for $(x,y) \in S(0,\hat{\alpha}_1,\hat{\alpha}_2,1)$. Let $P_1(1,1) = cL + \gamma$, where c is an integer

Otherwise, let $B(x,y) = P(x,y) - Q(x,y) = \frac{B_1(x,y)}{B_2(x,y)}$. Since $P \neq Q$ then $sign(B) \neq 0$, which implies that $sign(B_1) \neq 0$ and $sign(B_2) \neq 0$. By Lemma 2.2, $deg(B) \leq 2D$, and $max\text{-}coef(B) \leq 2(D+1)^2M^2$. Let $\alpha_1' = \max\{\ \alpha_1,\ \pi_1(B_1),\ \pi_1(B_2)\}$, and $\alpha_2' = \max\{\ \alpha_2,\ \pi_2(B_1),\ \pi_2(B_2)\}$. Lemma 2.3 gives the bounds $\alpha_1' \leq \max\{\alpha_1, 2D+1\}$ and $\alpha_2' \leq \max\{\alpha_1, 2(D+1)^2M^2+1\}$. Since the comparison $P(x,y) > Q(x,y)$ is equivalent to $B(x,y) > 0$, the lemma follows. □

The next step is showing the effect of rational operations on our parameters. This case is a rather simple case.

Lemma 4.4 *Let $P(x,y) = \frac{P_1(x,y)}{P_2(x,y)}$ and $Q(x,y) = \frac{Q_1(x,y)}{Q_2(x,y)}$ be two multivariate rational expressions, where $deg(P), deg(Q) \leq D$, and $max\text{-}coef(P)$, $max\text{-}coef(Q) \leq M$. Then, $deg(R \circ S) \leq 2D$, and $max\text{-}coef(R \circ S) \leq 2(1+D)^2M^2$, where $\circ \in \{+,-,*,/,\}$.*

Proof: A special case of Lemma 2.2. □

In the following lemma we show how to "handle" the floor operation, which is the most interesting case. As for one variable, the main objective is to find a rational expression that coincides with the value of the floor operation on some subset of the inputs. In order to get control over the degree and maximum coefficient of the rational expression we restrict the input to a subset of the previous inputs. The proof is similar in spirit to Lemma 3.4, and is also done cases. The cases represent different relationships between the lexicographic order of the two polynomials in the rational expression (to which the floor is applied). The main conceptual difference with the one variable proof technique is in Case 5 of the proof, where we introduce new variables.

Lemma 4.5 *Let $P(x,y) = \frac{P_1(x,y)}{P_2(x,y)}$ be a rational expression with integer coefficients, defined over $S(0,\alpha_1,\alpha_2,\alpha_3)$, such that $P(x,y) \geq 0$ for $(x,y) \in S(0,\alpha_1,\alpha_2,\alpha_3)$. Define* $\max\{4,\ \alpha_1,\ deg(P)\} = D$*, and* $\max\{4,\ \alpha_2,\ \alpha_3,\ max\text{-}coef(P)\} = M$*. Either*

The intuition is that the sequence $u_0, \ldots, u_{r+1}$ includes all the variables that we introduce. Notice that equations 4.1 and 4.2 imply that $u_0 > u_1 > u_2 > u_3 > \ldots > u_r > u_{r+1}$. The following definition relates the first pair (u_0, u_1), and the last pair (u_r, u_{r+1}).

Definition 7 *Let $S(r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda)$ denote the following set of ordered pairs of positive integers:*

$$\{(u_0, u_1) \ : \ \text{there exist integers } u_r, u_{r+1} \text{ such that } (u_0, u_1) \text{ is } <r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda> -\text{generated by } (u_r, u_{r+1})\}\,.$$

For convenience we omit the null vectors Δ and Λ whenever $r = 0$. In this case, the set $S(0, \alpha_1, \alpha_2, \alpha_3)$ consists of all pairs (u_0, u_1) such that $u_0 > (u_1)^{\alpha_1}$, $u_1 > \alpha_2$, $u_0 \equiv u_1 \equiv 1 \pmod{\alpha_3}$ (in accordance with the above definition).

Perhaps the most important characteristic of the above definitions is its similarity to the Euclidean algorithm for solving the gcd problem. As an immediate consequence of the definition, we get the two properties stated below. These properties are the key to our proof strategy.

Lemma 4.1 (The Correspondence Property): *There is an one-to-one correspondence between the elements of the sets $S(r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda)$ and $S(0, \alpha_1, \alpha_2, \alpha_3)$. Specifically, each pair $(u_0, u_1) \in S(r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda)$ corresponds to the unique pair $(u_r, u_{r+1}) \in S(0, \alpha_1, \alpha_2, \alpha_3)$ such that (u_r, u_{r+1}) is the $<r, \alpha_1, \alpha_2, \alpha_3, \Delta, \Lambda>$-generator of (u_0, u_1). Furthermore, if (u_0, u_1) corresponds to (u_r, u_{r+1}) then* $\gcd(u_0, u_1) = \gcd(u_r, u_{r+1})$.

Proof: Let $a_2 > a_3 > \ldots > a_{r+1}$ and $b_2 > b_3 > \ldots > b_{r+1}$, be the generating sequences for (a_0, a_1) and (b_0, b_1), respectively. It is easy to check that if $(a_i, a_{i+1}) \neq (b_i, b_{i+1})$ for some $0 \leq i \leq r$, then $(a_j, a_{j+1}) \neq (b_j, b_{j+1})$ for any j, $0 \leq j \leq r$. The assertion about the gcd's follows from the Euclidean algorithm. □

We give here a very brief description of the differences between the technique for two and one variables. The main difference between the two proof techniques is in the handling of the floor operation. Consider the expression $\left\lfloor \frac{y}{x} \right\rfloor$. If we assume that $x > y > 0$, then we can claim that the value of this floor operation is zero. On the other hand, consider $\left\lfloor \frac{x}{y} \right\rfloor$. Since we assume that $x > y > 0$, we have to find a new solution for such an expression. We overcome the problem by adding a new variable, z, and requiring that both $y > z > 0$, and $y + z = x$, which implies that $\left\lfloor \frac{x}{y} \right\rfloor = 1$. In some sense we transformed a computation based on the inputs x and y to a computation based on the input y and the "new input" z. The problem is that we introduce a new dependency between x and y, e.g. in our example $x = y + z < 2y$. The general solution has a similar flavor: we substitute x by $\lambda y^{\delta} + z$. The values of δ and λ are chosen to guarantee that previous requirements on the relationship between x and y hold.

The introduction of a new variable is the main difference between the technique for two variables and for one variable. Note that we may have to introduce many new variables, one for each floor operation. This causes additional complications in the proof technique that we need to resolve.

Using this proof technique we show a lower bound on the depth of any computation tree with $OP = \{+, -, *, /, \lfloor \cdot \rfloor\}$, that computes the gcd of all pairs of n-bit integers. The important part of the lower bound is that it is non-constant; quantitatively, the lower bound is $\Omega(\log \log n)$.

This chapter is organized as follows. In section 4.2 we describe the way that we structure the inputs that will be used in the proof. In section 4.3 we develop the proof technique. In section 4.4 we use the proof technique to show a lower bound for computing the greatest common divisor of two integers.

The following corollary states that for $M(n) = \Omega(2^{n^\epsilon})$, the lower bound is still $\Omega(\sqrt{\log n})$. This corollary is used later to derive an $\Omega(\sqrt{\log n})$ lower bound for various functions.

Corollary 3.7 *Let f be an $M(n)$-invariant function, such that $M(n) = \Omega(2^{n^\epsilon})$ for some fixed $\epsilon > 0$. Then any computation tree with $OP = \{+, -, *, /, \lfloor\cdot\rfloor\}$, that computes $f(x)$, for all n-bit integers, must have depth $\Omega(\sqrt{\log n})$.*

Using Corollary 3.7 we show lower bounds for the following problems.

1. Decide if $\lfloor \log a \rfloor$ is odd or even, for any n-bit integer a. (choose $M(n) = 2^{n-1}$, $a_1 = \lambda$ and $a_2 = 2\lambda$.)
2. Decide if $\lfloor \log\log a \rfloor$ is odd or even, for any n-bit integer a. (choose $M(n) = 2^{n/2}$, $a_1 = \lambda$ and $a_2 = \lambda^2$.)
3. Decide if $\sqrt{a}$ is an integer. (choose $M(n) = 2^{n/2-1}$, $a_1 = \lambda^2$ and $a_2 = 2\lambda^2$.)

Theorem 3.6 gives an $\Omega(\sqrt{\log n})$ lower bound on the depth of any decision tree with $OP = \{+, -, *, /, \lfloor\cdot\rfloor\}$, that solves the above problems.

At first sight it seems that all of the above lower bounds are for decision problems, and may be unrelated to the similar computation problems. In general this may be the case, but our examples have the property that the decision problem is closely related to the corresponding computational problem. For all the above decision problems, given a solution to the corresponding computation problem, one can decide the decision problem in $O(1)$ operations.

3.3 Applications

We start this section by characterizing a class of problems for which our technique gives non-trivial lower bounds. In the previous section we showed, that given a computation tree, we can find a path and a set of inputs $S(n, \lambda)$, such that all those inputs follow the path. The following defines an $M(n)$-invariant function to be one which has two inputs in $S(n, \lambda)$, for which the output is different.

Definition 5 *Let $f(x)$ be a boolean function whose domain is the set of n-bit integers, and let $M(n) > 0$. Then f is $M(n)$-invariant if, for any integer $\lambda < M(n)$, there are two n-bit integers a_1 and a_2, satisfying (i) $a_1 \equiv a_2 \equiv 0 \bmod \lambda$, and (ii) $f(a_1) \neq f(a_2)$.*

The following theorem states that the lower bound technique that we developed in the previous section gives a lower bound for any $M(n)$-invariant function.

Theorem 3.6 *Any decision tree with $OP = \{+, -, *, /, \lfloor \cdot \rfloor\}$ and constants $\{0, 1\}$, that computes an M(n)-invariant function, $f(x)$, for all n-bit integers, must have depth $\Omega(\sqrt{\log \log M(n)})$.*

Proof: Since f is a boolean function, we can assume that the leaves are labeled by either zero or one. (This adds at most two to the depth of the tree.) We prove the lower bound by contradiction. Suppose that we are given a decision tree T of depth $h < \frac{1}{2}\sqrt{\log \log M(n)}$. that decides f. In Theorem 3.5 we prove that there is a $\lambda \leq 2^{2^{4h^2}} < M(n)$, such that all the inputs in $S(n, \lambda)$ reach the same leaf. Since f is M(n)-invariant, there are a_1 and a_2, such that $a_1, a_2 \in S(n, \lambda)$ and $f(a_1) \neq f(a_2)$.

Both a_1 and a_2 follow the same path in T. Therefore T either accepts or rejects both $f(a_1)$ and $f(a_2)$. This contradicts the fact that T computes f. □

Clearly, the larger the value of M(n), the better the lower bound that we can show. On the other hand, since $M(n) \leq 2^n$, we can not show lower bounds greater than $\Omega(\sqrt{\log n})$.

This enables us to transform all the floor operations, on a specific path, to evaluations of rational expressions. Lemma 3.2 gives a way to restrict the input such that the comparison value is fixed. This will enable us to find an integer λ and a path $\mathcal{P}$, such that any integer in $S(n,\lambda)$ follows the path $\mathcal{P}$. Lemmas 3.2, 3.3, 3.4 also bound how fast the degree and the size of the coefficients of the rational expressions can grow. The bounds on the degree and maximum coefficients enable us to bound the value of λ, which is essential for showing the lower bound on complexity. For example, note that if $\lambda > 2^n$, then $S(n,\lambda) = \emptyset$, which would not enable us to derive any interesting lower bounds.

Theorem 3.5 *Let T be an decision tree of depth h with $OP = \{+,-,*,/,\lfloor\cdot\rfloor\}$. There is an integer $\lambda < 2^{2^{4h^2}}$ and a leaf v_l of T, such that for each input $a \in S(n,\lambda)$, the computation follows the path from the root of T to its leaf v_l.*

Proof: Denote the vertices on the path $\mathcal{P}$ from the root of T to its leaf v_l by $v_1, v_2, \ldots, v_l$, in that order, where v_1 is the root of the tree T and v_i is a child of v_{i-1}. We define the path $\mathcal{P}$ and the integer λ inductively, starting with the empty path and $\lambda^{(0)} = 1$. ($S(n,1)$ consists of all n-bit integers.) As part of the induction hypothesis, we maintain three properties of the path and the set under consideration. These properties are described below.

Suppose that (i) we have selected a prefix of $\mathcal{P}$, which starts at v_1, and ends at a vertex v_{i+1}, and (ii) defined a parameter $\lambda^{(i)}$ with the following properties:

1. For each input $a \in S(n,\lambda^{(i)})$ the computation follows the path from the root to v_{i+1};

2. For each computation vertex ν on the path from the root to the vertex v_{i+1}, excluding the vertex v_{i+1}, there is a pair of polynomials $(F_\nu(x), G_\nu(x))$ with integer coefficients, such that for each input $a \in S(n,\lambda^{(i)})$, $G_\nu(a) \neq 0$, and $val(\nu) = \frac{F_\nu(a)}{G_\nu(a)}$.

Case 3: $P_2(x) \prec P_1(x)$. Let L be the leading coefficient of $P_2(x)$. Note that since $sign(P_2) = +1$, then $L > 0$. Corollary 2.7 implies that

$$P_1(x) = \frac{A(x)P_2(x) + R(x)}{L^{D+1}},$$

where $A(x)$ and $R(x)$ are polynomials with integer coefficients such that, *max-coef*$(A) \leq 2^D M^{D+1}$, *max-coef*$(R) \leq 2^{D+1} M^{D+2}$, $deg(A) \leq D$ and $deg(R) < D$.

Consider the constant term of $A(x)$. We denote this constant by $cL^d + \gamma$, where c is an integer and γ is a non-negative integer such that $0 \leq \gamma < L^d$. Let $A(x) = \hat{A}(x) + cL^d + \gamma$; that is, $\hat{A}(x)$ is equal to the polynomial $A(x)$ minus its constant term. The parameter π will be chosen to be a multiple of L^d. This implies that for each $a \in S(n, \lambda\pi)$, each monomial of $L^{-d}\hat{A}(a)$ evaluates to an integer. Hence, for each integer $a \in S(n, \lambda\pi)$,

$$\left\lfloor \frac{P_1(a)}{P_2(a)} \right\rfloor = \frac{1}{L^d}\hat{A}(a) + c + \left\lfloor \frac{\gamma P_2(a) + R(a)}{L^d P_2(a)} \right\rfloor .$$

We distinguish between two subcases:

Subcase 1: $\gamma > 0$. Consider the polynomial $V(x) = L^{-d}(\gamma P_2(x) + R(x))$. The leading coefficient of V is $L^{-d}(\gamma L)$, therefore $sign(V) = +1$. Let $B(x) = P_2(x) - V(x)$. The leading coefficient of $B(x)$ is $(1 - L^{-d}\gamma)L$. Since $0 < L^{-d}\gamma < 1$, the leading coefficient of $B(x)$ is positive, i.e. $sign(B) = +1$. Using Corollary 2.4, let π be the minimum multiple of L^d such that $\pi \geq \max\{\pi(B), \pi(\gamma P_2 + R), \pi(P_1), \pi(P_2)\}$. Consider $a \in S(n, \lambda\pi)$. Since $\pi \geq \pi(B)$, then $V(a) < P_2(a)$. Since $\pi \geq \pi(\gamma P_2 + R)$, then $0 < V(a)$, which implies that $0 < V(a) < P_2(a)$. Since $\pi \geq \pi(P_2)$, then $P_2(a) > 0$. This implies that $0 < \frac{V(a)}{P_2(a)} = \frac{\gamma P_2(a) + R(a)}{L^d P_2(a)} < 1$, for any $a > \pi$. We conclude that for each $a \in S(n, \lambda\pi)$, $\left\lfloor \frac{P_1(a)}{P_2(a)} \right\rfloor = L^{-d}\hat{A}(a) + c$. Let

$$Q(x) = \frac{\hat{A}(x) + cL^d}{L^d} = \lfloor P(x) \rfloor \quad \text{for } x \in S(n, \lambda\pi).$$

The following claim shows how we can restrict the input by changing the parameter of $S(\cdot)$. Namely, multiplying the parameter by another integer and considering only integers that are divisible by the product.

Claim: 3.1 *For any integer $\pi \geq 1$, $S(n, \lambda\pi) \subset S(n, \lambda)$.*

The following lemma enables us, each time the computation reaches a comparison vertex, to fix one of the directions. This involves restricting the inputs, from $S(n, \lambda)$ to a subset of it, $S(n, \lambda\pi)$. An important part of the lemma is to bound the value of π.

Lemma 3.2 *Let $P(x)$ and $Q(x)$ be two rational expression, of degree at most D and maximum coefficient at most M, defined over $S(n, \lambda)$. There exists $\pi \leq 2(1 + D)M^2 + 1$, such that for all $x \in S(n, \lambda\pi)$, the comparison $P(x) \geq Q(x)$ is determined (i.e. either $P(x) = Q(x)$, $P(x) > Q(x)$, or $P(x) < Q(x)$, for $x \in S(n, \lambda\pi)$).*

Proof: Let $B(x) = P(x) - Q(x) = \frac{B_1(x)}{B_2(x)}$. If $P(x) = Q(x)$ for $x \in S(n, \lambda)$, let $\pi = 1$, and the lemma follows. Otherwise both $sign(B_1) \neq 0$ and $sign(B_2) \neq 0$. By Lemma 2.2 the degree of B is at most $2D$ and *max-coef*$(B) \leq 2(D + 1)M^2$.

Corollary 2.4 guarantees the existence of a positive integer $\pi = \max\{\pi(B_1), \pi(B_2)\}$, such that for all integers $x > \pi$, either $B(x) > 0$, or $B(x) < 0$. Furthermore, it guarantees that $\pi \leq 2(D + 1)M^2 + 1$. Since the expression $P(x) > Q(x)$ is equivalent to $B(x) > 0$, the lemma follows. □

Next, we consider the case of a computation vertex. First we discuss rational operations, and restate Lemma 2.2, for univariate rational expressions.

Lemma 3.3 *Let P and Q be two rational expressions, of degree at most D and maximum coefficient at most M, defined over $S(n, \lambda)$. Let $R(x) = P(x) \circ Q(x)$, where $\circ \in \{+, -, *, /\}$. Then, deg$(R) \leq 2D$ and max-coef$(R) \leq 2(D + 1)M^2$.*

Essentially, the degree argument uses the fact that a polynomial cannot change "too fast". Namely, a polynomial of degree d can change its sign at most $d+1$ times. Therefore a polynomial of degree d can "disconnect" the inputs to at most $d+1$ different connected components. The number of connected components of the function to be computed implies a lower bound on the degree of the polynomial, which implies a lower bound on the depth.

The degree argument can be extended to rational functions, and also in higher dimensions. The arguments for higher dimensions are much more complex than for one dimension, and they are the main part of the lower bound technique developed in [BO83, SY82, Yao89].

Adding the `floor` function as a basic operation changes the situation dramatically. First, using the floor function, one can trivially compute the parity function in $O(1)$ steps. Second, the floor function may increase the number of connected components in an unbounded way. For this reason the number of connected components is not relevant in computations with the floor function.

In the remainder of this section we give a general overview of the proof technique that we develop. Given a decision tree, the proof constructs a path from the root to a leaf, such that a certain set of inputs follows that path. The property of the set, is that it includes all the integers that are divisible by a certain integer λ. A main part of the proof technique is to bound λ as a function of the depth of the decision tree, i.e. to show that $\lambda \leq g(h)$, where h is the depth of the tree. In Section 3.3 we show applications of this proof technique. We define a boolean function to be $M(n)$-invariant if for *any* integer $\lambda < M(n)$, there are at least two n bit integers, both divisible by λ, for which the value of the function differs. We argue that any $M(n)$-invariant boolean function requires depth at least h, where $g(h) = M(n)$.

As one can expect, the main part of the proof technique is devoted to handling the floor function. The main idea is to transform the floor function to a rational function.

For univariate polynomial the situation is even simpler, and is summarized in the following corollary of Lemma 2.5.

Corollary 2.7 *Let $P(x)$ and $Q(x)$ be two polynomials with integer coefficients, of degree at most D, and maximum coefficient at most M. Let L be the leading coefficient of $Q(x)$, then there exists $A(x)$ and $R(x)$ such that,*

$$P(x) = \frac{1}{L^{D+1}} A(x)Q(x) + \frac{1}{L^{D+1}} R(x),$$

where $A(x)$ and $R(x)$ are polynomials with integer coefficients, and max-coef$(A) \leq 2^D M^{D+1}$, max-coef$(R) \leq 2^{D+1} M^{D+2}$, deg$(A) \leq D$, and deg$(R) < D$.

$R(x,y)$ such that

$$P(x,y) = \frac{1}{L^{\delta+1}}A(x,y)Q(x,y) + \frac{1}{L^{\delta+1}}R(x,y),$$

where $A(x,y)$ and $R(x,y)$ are polynomials with integer coefficients. Furthermore,

1. *max-coef$(R) \leq (2 + deg_y(Q))^{\delta+1} M N^{\delta+1}$,*
2. *$deg_x(R) < deg_x(Q)\{= d\}$, and $deg_y(R) \leq deg_y(P) + (\delta+1)deg_y(Q)$.*
3. *$deg_x(A) \leq \max\{0, \delta\}$, and $deg_y(A) \leq deg_y(P)$, and*
4. *max-coef$(A) \leq (2 + deg_y(Q))^{\delta} M N^{\delta}$.*

Proof: The proof is by induction on δ. The hypothesis holds for the basis case $\delta = -1$ with $A(x,y) = 0$ and $R(x,y) = P(x,y)$.

For the induction step, assume that the hypothesis holds for all $\delta < k$, for some $k > -1$. We prove it for k. Let $P(x,y) = p_1(y)x^e + p_2(y)x^{e-1} + \ldots$, be such that $k = e - d$. Consider the polynomial

$$S(x,y) = LP(x,y) - x^k p_1(y)Q(x,y).$$

One can verify that, *max-coef$(S) \leq (2 + deg_y(Q))MN$*, $deg_x(S) \leq deg_x(P) - 1$, and $deg_y(S) \leq deg_y(P) + deg_y(Q)$.

Applying the hypothesis to the pair $S(x,y)$ and $Q(x,y)$, yields

$$S(x,y) = \frac{1}{L^{\delta}}A'(x,y)Q(x,y) + \frac{1}{L^{\delta}}R'(x,y).$$

Substituting for $S(x,y)$, we get

$$P(x,y) = \frac{1}{L^{\delta+1}}\left(A'(x,y) + L^{\delta}x^k p_1(y)\right)Q(x,y) + \frac{1}{L^{\delta+1}}R'(x,y).$$

Bivariate and univariate polynomials

A *bivariate polynomial* is a polynomial with two variables. Below, we relate the lexicographic order defined on the bivariate polynomials, and the order among their values at certain points. We concentrate on bivariate and univariate polynomials since those are the cases that we would be interested in later.

Consider a bivariate polynomial $P(x,y)$. We would like to give a simple sufficient condition on the inputs of P, such that for any input (x,y) that satisfies the condition, the sign of $P(x,y)$ would be the same as $sign(P)$. The sufficient condition that we give in the lemma below guarantees that the value of the leading monomial is larger than all other monomials combined. This is clearly sufficient to ensure that $sign(P(a,b)) = sign(P)$, for such inputs (a,b).

Lemma 2.3 *For each bivariate polynomial $P(x,y)$, such that $sign(P) \neq 0$, there exist positive integers $\pi_1(P)$ and $\pi_2(P)$ such that for all (a,b) satisfying $a > b^{\pi_1(P)}$ and $b > \pi_2(P)$, $sign(P(a,b))$ equals $sign(P)$. Furthermore, $\pi_1(P) \leq deg_y(P)+1$, and $\pi_2(P) \leq \frac{max\text{-}coef(P)+|L|}{|L|}$, where L is the leading coefficient of P.*

Proof: Let $M = max\text{-}coef(P)$. Let $P(x,y) = \sum_{k=0}^{m} L_k x^{i_k} y^{j_k}$, where $P(x,y)$ is written in its normal form, and L_k is the coefficient of the k^{th} monomial ($L_0 = L$). Note that since $sign(P) \neq 0$, then $L \neq 0$. Denote $t_k(x,y) = x^{i_k} y^{j_k}$. Let $\pi_1(P) = 1+\max_{0\leq k\leq m-1}\{0, j_{k+1} - j_k\}$, and $\pi_2(P) = \frac{M+L}{L}$. Clearly, $\pi_1(P) \leq deg_y(P) + 1$.

From the lexicographic order it follows that $\frac{t_{k+1}(x,y)}{t_k(x,y)} = x^i y^j$, where either $i < 0$ *and* $j < \pi_1(P)$ or $i = 0$ *and* $j < 0$. Thus, if $a > b^{\pi_1(P)}$ then $\frac{t_{k+1}(a,b)}{t_k(a,b)} < \frac{1}{b}$, and hence, $\frac{t_k(a,b)}{t_0(a,b)} < \frac{1}{b^k}$.

Suppose that $L > 0$, i.e. $sign(P) = +1$. We show that for (a,b), satisfying $a > b^{\pi_1(P)}$ and $b > \pi_2(P)$, $P(a,b) > 0$, i.e. $sign(P(a,b)) = +1$. For (a,b), such that $a > b^{\pi_1(P)}$,

$$P(a,b) \geq L\, t_0(a,b) - \sum_{k=1}^{m} |L_k| a^{i_k} b^{j_k} > L\, t_0(a,b)(1 - \sum_{k=1}^{m} \frac{|L_k|}{Lb^k}).$$

Proof: We only prove the bound on *max-coef*$(P*Q)$. (The other bounds are straightforward.) Note that a multivariate polynomial of degree d has at most $(1+d)^k$ monomials. Therefore, when we multiply two multivariate polynomials, each of the coefficients of the product is the sum of at most $(1+\min\{deg(P), deg(Q)\})^k$ terms, each of them being the product of a coefficient in P by a coefficient in Q. Since the product of a coefficient of P and a coefficient of Q is bounded by *max-coef*(P) *max-coef*(Q), the bound on *max-coef*$(P*Q)$ follows. □

A *rational number* is a number that can be expressed as n/m, where n and m are integers. Similarly, a *rational expression* $R(x_1,\ldots,x_k)$ can be written as $\frac{P(x_1,\ldots,x_k)}{Q(x_1,\ldots,x_k)}$, where P and Q are polynomials.

For a rational expression $R(x_1,\ldots,x_k) = \frac{P(x_1,\ldots,x_k)}{Q(x_1,\ldots,x_k)}$, where P and Q are multivariate polynomials, define the *degree* of R to be the larger of the degrees of P and Q. Similarly, define the *maximum coefficient* of R to be the larger of the maximum coefficients of P and Q.

Lemma 2.2 *Let* $R(x_1,\ldots,x_k) = \frac{P_1(x_1,\ldots,x_k)}{P_2(x_1,\ldots,x_k)}$ *and* $S(x_1,\ldots,x_k) = \frac{Q_1(x_1,\ldots,x_k)}{Q_2(x_1,\ldots,x_k)}$ *be two multivariate rational expressions. Then,*

1. $deg(R \pm S) \leq deg(R) + deg(S)$,
2. $max\text{-}coef(R \pm S) \leq 2(1+\min\{deg(R), deg(S)\})^k max\text{-}coef(R) max\text{-}coef(S)$,
3. $deg(R * S) \leq deg(R) + deg(S)$, *and*
4. $max\text{-}coef(R * S) \leq (1+\min\{deg(R), deg(S)\})^k max\text{-}coef(R) max\text{-}coef(S)$.

Proof: The bounds follow from Lemma 2.1 and from the fact that for $\circ \in \{+,-\}$, $R \circ S = \frac{P_1 Q_2 \circ Q_1 P_2}{P_2 Q_2}$. □

The computation terminates at a leaf u and outputs the value of u. The *time complexity* of a given input is the length of the path that the computation traverses using that input. The *time complexity of a computation tree* is the maximum time complexity over all inputs, which is equal to the *depth* of the tree[1], assuming that each vertex can be reached by some computation.

As one can note, we did not mention how constants appear in the tree. We can introduce constants as additional operations, and view a constant as a constant function. In many cases it is simpler to state separately the basic operations and the allowed constants. However, the formal interpretation of such a statement is that there is one set of basic operations, that includes the constants and basic operations.

A *decision tree* is a computation tree whose output values are either 0 or 1. For decision trees we can assume that each leaf is labeled by a constant, either 0 or 1. This can increase the depth of the tree by an additive factor of at most two.

A *straight line program* is a computation tree that does not include any comparison vertices.

The operations $\{+,-,*\}$ are defined in the natural way. The rational division operation returns a rational function of the inputs, e.g. $2/3 = 0.666\cdots$. (In case a division by zero occurs during a computation, the output of the entire computation is undefined.) The set of *rational operations* includes the operations $\{+,-,*,/\}$. We define the floor operation in the following way. The *floor operation* receives as an operand a non-negative real number and returns the largest integer smaller than or equal to the operand.

Our assumption that the operand to the floor function is non-negative simplifies our proofs and does not change the depth of the computation tree by more than a constant factor. Using this floor operation, one can implement either a general floor operation (i.e. the operand is an arbitrary real number) or a mod operation using $O(1)$ depth.

[1] The depth of a tree is the length of a longest path from the root to a leaf.

However, this is not the case in general. In most cases we would desire some conditional branching mechanism. The time complexity of a program that includes such a mechanism cannot be estimated by the number of instructions that appear in the program. This is best illustrated in the following example. Consider the following program:

```
$z := 0$
WHILE $y \geq 1$ DO
        $z := z + x$
        $y := y - 1$
END WHILE
```

For $y \in [1, n]$, the complexity of the above procedure is $O(n)$ operations. The following program, that looks very similar requires only $O(\log n)$ operations.

```
$z := 0$
WHILE $y \geq 1$ DO
        $z := z + x$
        $y := y/2$
END WHILE
```

In order to avoid such problems, we "unroll" the loop structure. This means that the program is represented as a tree, possibly an infinite tree. The time complexity of a given input is the length of the path in the tree traversed by the input. We would consider only finite trees.

After the above motivation, we can give a precise definition of the computation tree model.

Unlike much of the previous research, our work concentrates on functions with a constant number of input variables. This allows us to get a better understanding of the complexities that are involved with the floor operation. Since there are only a constant number of inputs, we define the complexity measures as a function of the complexity of the input, e.g., the number of bits that represent an integer input. We develop a general technique for deriving lower bounds for such functions. In combination with the lower bounds, we show upper bounds that use the floor function in new and novel ways. In some cases we are able to derive matching lower and upper bound, up to a constant multiplicative factor.

We develop a general technique to handle functions of a single n-bit integer input. Using this technique we derive $\Omega(\sqrt{\log n})$ lower bounds for problems such as computing $\lfloor \log\log x \rfloor$, deciding if the input is a perfect square (i.e. whether its square root is also an integer), and other problems. For computing $\lfloor \log\log x \rfloor$, we give a $O(\sqrt{\log n})$ algorithm, and thus show that this lower bound is tight. (Note that using only rational operations, computing $\lfloor \log\log x \rfloor$ requires $\Theta(\log n)$ operations.)

We extend this technique to handle two variable functions, and show that computing the greatest common divisor of two integers requires a non-constant number of operations. More precisely, we show an $\Omega(\log\log n)$ lower bound for computing the greatest common divisor of two n-bit integers. This lower bound holds even if we restrict our attention to deciding if a pair of integers are relatively prime or not.

We also develop a lower bound technique for approximating real valued functions. We consider approximating the s^{th} root of a real number from the interval $[1,2]$, on the "average", where s is a fixed constant. We define an average approximation by considering the function which gives the absolute difference between the approximated value and the correct value, for each $x \in [1,2]$, and integrating this function over the interval $[1,2]$. The value of this integral is the "average error" of the approximation. We show that in order to approximate the s^{th} root of an input from $[1,2]$, such that

At the beginning of this decade, a general technique, based on topological arguments, was developed in [BO83, SY82]. The technique relates the logarithm of the number of connected components of a decision problem and the number of rational operations the problem requires. Based on this technique, many problems, whose inputs are either real or rational numbers, have been given tight bounds. This technique, under some restrictions, was extended to handle integral input by [Yao89].

Another model that uses abstract arithmetic operations, rather than explicit bit operations, is the *strongly polynomial* model. The motivation for this model is to achieve polynomial time algorithms that would not depend on the representation of the input. This would allow inputs with infinite representation, as in the case of real numbers. The aim is that the number of operations would be polynomial in the number of inputs, and not depend on the size of the input. Grötschel, Lovász, and Schrijver, in their book "Geometric Algorithms and Combinatorial Optimization", asked if there exists a strongly polynomial algorithm for the greatest common divisor of two integers. (See [GLS88], pp. 32-33, p. 225.) Notice that since there are only two integer inputs to the problem, any strongly polynomial algorithm for this problem must have a constant number of arithmetic operations.

There are two "standard" sets of operations that are considered in strongly polynomial computations. The weaker model considers only rational operations, i.e. $\{+, -, *, /\}$. The more powerful model has an additional "rounding down" operation (i.e. floor). Stockmeyer [Sto76] proves an $\Omega(n)$ lower bound for deciding if an n-bit integer is odd or even, in the weaker model, where only rational operations are allowed. We show in this work that there is no strongly polynomial algorithm for the greatest common divisor problem in the stronger model, where rounding operations are allowed. Our result implies that strongly polynomial time is different from polynomial time.

When defining strongly polynomial computations one normally restricts the size of the integers involved in the computation to have size polynomial in the size of the original

I would like to thank Mike Sipser for discussion that we had in the early stages of the research, discussion that helped me to define the problems in the early stages of the research. I would like to thank Larry Stockmeyer who served on my thesis committee. His numerous comments about this thesis have greatly to improve its presentation.

I would like to thank the many friends I had in the theory group, especially Mark Newman, Noam Nisan, Phill Rogaway, Arie Rudich and Nir Shavit. I would like to thank all the theory group for the unique atmosphere that makes the research process a pleasure.

I would like to thank Nina Wiener and ISEF for both their support and assistance during my graduate study, help which is well appreciated by me.

I am grateful to IBM where I spent two summer, and for the financial support I received through the IBM graduate student fellowship. I would like to thank David Johnson and AT&T, for the summer I spent there.

A special thanks to my parents Aviva and Jacob, for their continuous love, support and encouragement during all the many years of my studies. I would also like to thank Taly, Ido, and Yuval, for their love and encouragement.

The person that I am most in debt to for helping me through my graduate years is my spouse Lea. Her love and encouragement have always helped me to bounce back to the right track. There are no words that can express how important her support has been for me.

common divisor of two integers requires a non-constant number of operations; such a lower bound would separate polynomial time and strongly polynomial time.

We resolve this open problem and show that there is no computation tree that computes the `greatest common divisor` in a fixed number of operations; in fact, we prove an $\Omega(\log\log n)$ lower bound for computing the greatest common divisor of two n-bit integers. Thus separating polynomial time and strongly polynomial time.

3. *Approximation problems.* In this category we consider the complexity of computing an approximation to the s^{th} root of a real number. The notion of approximation that we consider is "on the average", which is modeled by an L_1 norm. We show a $\Omega(\sqrt{\log\log\frac{1}{\epsilon}})$ lower bound for approximating the s^{th} root of inputs in the interval $[1,2]$, within ϵ on the average. (The inputs to the computation, in this case, is the number for which the s^{th} root is approximated.)

   We also show interesting upper bounds. Using Newton's method one can approximate the s^{th} root in $O(\log\log\frac{1}{\epsilon})$ rational operations, and this bound is tight (for rational operations). Using the floor function, one can accelerate the computation, and compute an approximation in $\Omega(\sqrt{\log\log\frac{1}{\epsilon}})$ operations. (The input to the computation, in this case, are both the number for which the s^{th} root is approximated, and the error parameter ϵ.)

Thesis Supervisors:
Shafi Goldwasser, Associate professor of Computer Science and Engineering.
Baruch Awerbuch, Associate professor of Mathematics.