Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Kristina Hostáková

The BSS model and cryptography

Department of Algebra

Supervisor of the master thesis:	prof. RNDr. Jan Krajíček, DrSc.
Study programme:	Mathematics
Study branch:	Mathematical Methods of Information Security

Prague 2016

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague on 11 May 2016 Kristina Hostáková

Title: The BSS model and cryptography

Author: Kristina Hostáková

Department: Department of Algebra

Supervisor: prof. RNDr. Jan Krajíček, DrSc., Department of Algebra

Abstract: Real numbers are usually represented by various discrete objects such as floating points or partial decimal expansions. This is mainly because the classical computability theory relates to computers which work with discrete data. Nevertheless, for theoretical purposes it is interesting to look at models of computation that deal with real numbers as with objects of unit size. A very natural such model was suggested by Blum, Shub and Smale in 1989.

In 2012 Grigoriev and Nikolenko studied various cryptographic tasks involving real numbers (for example, biometric authentication) and they considered the BSS machine model. In this work we focus on hard to invert functions in this model of computation. Our main theme is to analyse whether there are real functions of one variable that are easier to compute than to invert by a BSS machine.

Keywords: real computation, BSS machine, computable function, hard to invert function

Název práce: BSS model a kryptografie

Autor: Kristina Hostáková

Katedra: Katedra algebry

Vedoucí diplomové práce: prof. RNDr. Jan Krajíček, DrSc., Katedra algebry

Abstrakt: Reálná čísla jsou obvykle reprezentována různými diskrétními objekty, například plovoucí řádovou čárkou nebo částečným desetinným rozvojem. A to zejména proto, že se klasická teorie vyčíslitelnosti váže k počítačům, které pracují s diskrétními daty. Nicméně z teoretického hlediska je zajímavé uvažovat i výpočetní modely, které pracují s reálnými čísly jako s objekty velikosti jedna. Blum, Shub a Smale v roce 1989 takový model navrhli.

V roce 2012 se Grigoriev a Nikolenko zabývali různými kryptografickými úlohami, které se přirozeně týkají reálných čísel (například biometrickou autentizací) a uvažovali při tom výpočetní model BSS. V této práci se zaměřujeme na těžko invertovatelné funkce v tomto výpočetním modelu. Naším hlavním cílem je zkoumat, zda existují reálné funkce jedné proměnné, které se BSS strojem snáze počítají než invertují.

Klíčová slova: počítání s reálnými čísly, BSS stroj, vyčíslitelná funkce, těžko invertovatelná funkce

Firstly, I would like to thank my supervisor, prof. RNDr. Jan Krajíček, DrSc., for his guidance. I am most grateful for all the time he spent with me during many consultations and for his suggestions and valuable comments. His positive attitude was very inspiring and motivated me during the work on this thesis.

I would also like to thank my family and closest friends for their unconditional support.

Contents

Introduction

There are two main approaches how to model computations with real numbers. In the first model, which is usually called the bit-model of real computation, we treat real numbers as discrete objects. This is, indeed, a very realistic approach since computations in the everyday life are done by computers, which work with binary data. Thus, they can store a real number only with a finite precision.

In this work we consider the second approach of modelling computations with real numbers, which is sometime called the algebraic approach. In contrast to the bit-model, elements of $\mathbb{R}$ are now viewed as unit size objects which can be stored with infinite precision. In addition, basic arithmetic operations with real numbers, such as addition, subtraction, multiplication and division, are considered to be unit time operations. Moreover, we can compare two real numbers in a unit of time in this model. This abstract approach preserves the continuity property of real numbers and authentically reflects "the difficulty" of computing a real function from the theoretical point of view. On the other hand, let us emphasis that this approach does not capture the everyday life computations performed by computers and is investigated for theoretical purposes.

These two directions are, indeed, very different. Their relationship is studied in detail for example in [3].

Several algebraic models of real computation have been presented in the literature. An algorithm computing with real numbers can be formally described for example by an algebraic circuit or a computation tree. An overview of different models of computation can be found in [4].

In this thesis we consider probably the most universal algebraic model of real computation which was presented by Lenore Blum, Michael Shub and Steve Smale in [2]. It was named after its inventors: the Blum-Shub-Smale model or briefly **the BSS model**. Analogously to the Turing model of computation, algorithms are formalized as machines which are called Blum-Shub-Smale machines or briefly **the BSS machines**. They can be defined over an arbitrary commutative ring with unit (for example over $\mathbb{R}, \mathbb{C}, \mathbb{Q}, \mathbb{Z}$ or $\mathbb{Z}_2$). The BSS model over $\mathbb{Z}_2$ is equivalent to the classical Turing model of computation. In addition, ordinary Turing machines can be simulated by BSS machines over any ring with unit. However, it is commonly expected that Turing machines cannot simulate BSS machines over real numbers (not even if the inputs are only integers). We briefly discuss in Chapter 5 why we agree with this conjecture. Due to this relation between BSS and Turing machines, the BSS model over $\mathbb{R}$ is sometimes also called the "real" Turing model.

In the book [1] Blum, Cucker, Shub and Smale showed that many concepts of classical computational complexity theory can be naturally extended to the BSS model of computation. For instance, computable function, decidable set or complexity classes P and NP can be defined with respect to an arbitrary commutative ring with unit in this model. The advantage of BSS machines is that, although they are able to capture infinite computations, machines themselves are finite objects. Moreover, BSS machines can be describe algebraically. In particular, halting sets of BSS machines are countable unions of semi-algebraic sets and input-output maps of BSS machines are piecewise rational maps.

A fundamental object in the classical computability theory is a one-way function. It is a famous open problem whether such a function exits. It is unclear how to even define a continuous analogy of this basic cryptographic primitive. There appear several possible definitions in the literature; however, none of them is widely accepted. Hard to invert continuous functions were studied for example by Grigoriev and Nikolenko. The candidates which they suggested in [7] are based on tropical and super-tropical circuits. Another point of view on cryptographic primitives in an unbounded computation model can be found in [15].

One reason why it is interesting to study continuous hard to invert functions is biometric authentication. We have a biological trait which changes a little over time. The goal is to accept the traits which is "very close" to the original sample and reject all others. There are several ways how to deal with this problem in practice. A trait is usually described by a set of discrete patters which is then compared with the original set of patters. Thus, the continuity is understood in a discrete sense. Although real models of computation are not realistically modelling everyday life computations, they are able to capture the continuity property of the biometric authentication in a very natural way.

The main theme of this work is to study real functions which are "easy to compute but hard to invert" in the BSS model of computation. Our aim is to first formalize what is meant by "easy to compute" in the BSS model and analyse which real functions satisfy our definition. Then we consider two possible ways how to understand the term "invert" in the context of real numbers. The first approach follows the classical model of computation and requires inversion with infinite precision. By this we mean that, given a value $f(x)$, the task is to find an x' such that $f(x') = f(x)$. The second approach corresponds better with the motivation from biometric authentication since it reflects the continuity property of real numbers. It allows inversion with an arbitrary but fixed precision $\epsilon > 0$. This means, given a value $f(x)$, we have to find an input x' such that $|f(x) - f(x')| < \epsilon$. And finally, we discuss (from both points of view) the time complexity of a BSS machine which inverts an easy to compute real function. This is mainly done in Chapter 4. The organization of the rest of this thesis is the following.

In Chapter 1 we establish the terminology and the notation used throughout this work. Then we prove several technical statements about sets and rational functions over real numbers which are useful in later chapters.

In Chapter 2 we introduce the BSS model of computation, explain the idea of BSS machines on finite dimensional BSS machines and discuss the nice algebraic properties of BSS machines. We also present an alternative point of view on BSS machines which provides a useful framework for a detailed complexity analysis.

In Chapter 3 we study machines that compute the square root function $y \mapsto \sqrt{y}$. Or equivalently we can say that we focus on machines that invert the square function $x \mapsto x^2$. We first prove that no BSS machine can compute the square root function with infinite precision. Thereafter, we construct two BSS machines which approximate the square root function with some fixed precision $\epsilon > 0$. The time complexity of the first machine is $\mathcal{O}(\log_2(y) - \log_2(\epsilon))$. The second machine is more sophisticated. It uses Newton's method and we prove that its time complexity is $\mathcal{O}\left((\log_2(\log_a(y) - \log_a(\epsilon)))^2\right)$, where $a \approx 1.5625$. Moreover, we also discuss a lower bound on the time complexity of inverting x^2 with an arbitrary but fixed precision $\epsilon > 0$. Our bound is of the following form. A BSS machine inverting x^2 on the interval $[0, \sqrt{N}]$, for some $N \in \mathbb{N}$, has to either perform $\Omega(\log_2(\log_2(N) - \log_2(\epsilon)))$ elementary operations or use more machine constants than just 0 and 1.

And finally, in Chapter 5 we present some possible extensions of this work.

Before we proceed to the first chapter of this thesis, let us make two comments. Chapters 3 and 4 discuss various algorithms described in the BSS model and this is often accompanied by a number of detailed but elementary computations; this is done in order to analyse the time complexity of the algorithms. We would also like to emphasis that in Chapter 2 we present concepts and results from the book [1] and not our work.

1. Preliminaries

In this chapter we first establish the terminology and notation used throughout this thesis. In Section 1.2 we focus on sets of real numbers. We introduce some less common notation and state technical lemmas which will be useful in later chapters. Finally, in Section 1.3 we briefly discuss what we know about polynomials and rational functions over $\mathbb{R}$. Moreover, we prove several auxiliary statements needed further in the work.

In this thesis we assume basic knowledge of algebra, mathematical analysis, set theory, complexity and computability theory in the Turing model of computation.

1.1 Terminology and notation

We use the standard notation for the most common sets of numbers which are $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$. Let us specify that we do not consider zero to be a natural number; this means that $\mathbb{N} = \{1, 2, 3, \dots\}$. In this work $\mathbb{R}^+$ denotes the set of all positive real numbers and $\mathbb{R}^{\geq 0}$ denotes the set of all non-negative real numbers.

By an interval with endpoints $a \in \mathbb{R} \cup \{-\infty\}, b \in \mathbb{R} \cup \{\infty\}$ we understand either an open interval (a, b), a half-closed interval $(a, b]$, a half-closed interval $[a, b)$ or a closed interval $[a, b]$. As usual, the round bracket indicates that the endpoint is not an element of the interval whereas the square bracket denotes the opposite. We call an interval non-degenerated (or non-trivial) if $a < b$. An interval with endpoint a, b is unbounded if $a = -\infty$ or $b = \infty$. A non-degenerated interval is bounded if its endpoints satisfy the inequalities

$$-\infty < a < b < \infty.$$

By a sequence we mean an ordered list of objects of the same kind. Repetitions are allowed. We consider both finite and infinite sequences in this work. The following two examples establish the notation. The sequence

$$\left(\frac{1}{2}, \sqrt{2}, \pi, \sqrt{2}, 39.09\right)$$

is an example of a finite sequence of real numbers and the sequence

$$\left(2^i\right)_{i \in \mathbb{N}}$$

is an example of an infinite sequence of natural numbers. It should be always clear from the context whether (a, b) represents an open interval or a finite sequence of length two.

Sometimes we will refer to the Turing model of computation as to "the classical model of computation". We do not define basic notions, such as Turing machine, one-way function, one-way permutation, complexity classes P and NP and so forth. However, these definitions can be found in any textbook about complexity and computability theory in the Turing model of computation. See, for example, the book [10] written by M. Sipser or the book [8] whose author is C. H. Papadimitriou.

To measure the complexity of algorithms, we use the standard asymptotic notation. Let $f, g\colon \mathbb{N} \to \mathbb{R}^+$ be two functions. We write $f(n) = \mathcal{O}(g(n))$ if there exist $c, n_0 \in \mathbb{N}$ such that for every $n \geq n_0$ the following inequality holds

$$f(n) \leq c \cdot g(n).$$

This notation is sometimes called the "Big-$\mathcal{O}$" notation and it is used for asymptotic upper bounds. In this work we are also interested in lower bounds for which we use the "Big-Ω" notation. We write $f(n) = \Omega(g(n))$ if there exist $c, n_0 \in \mathbb{N}$ such that for every $n \geq n_0$ the following inequality holds

$$f(n) \geq c \cdot g(n).$$

For more details about asymptotic notation see one of the books [8] and [10].

As already mentioned, we assume basic knowledge of real mathematical analysis. We will use but not define notions such as continuous function, limit of a real function, point of global extreme of a real function, derivation of a real function and so on. We will refer to some important theorems, for instance, to the "Intermediate Value Theorem" or to the "Algebraic Limit Theorem". The statements of these theorems can be found in any textbook of real mathematical analysis, for example, in the book [9].

1.2 Sets of real numbers

In this section we first introduce basic arithmetic operations on subsets of $\mathbb{R}$. Then we recall the definition and main properties of semi-algebraic sets since sets defined by polynomial equalities and inequalities will be very important in future chapters of this work.

1.2.1 Basic arithmetic operations with sets

To avoid a confusion, sets will be denoted by bold capital letters. Throughout this thesis, we consider only sets of real numbers containing 0 and 1 (if not explicitly said otherwise). The reason of this convention is technical and will become more clear later (see Observation 1.2).

Let us first define basic arithmetic operations on subsets of $\mathbb{R}$.

Definition 1.1. Let $\mathbf{A}, \mathbf{B} \subseteq \mathbb{R}$. Then we define binary operations $\cdot$ and $+$ as follows

$$\begin{aligned}\mathbf{A} + \mathbf{B} &= \{a + b\colon a \in \mathbf{A}, b \in \mathbf{B}\},\\ \mathbf{A} \cdot \mathbf{B} &= \{a \cdot b\colon a \in \mathbf{A}, b \in \mathbf{B}\}.\end{aligned}$$

For $t \in \mathbb{N}$ we denote

$$t\mathbf{A} = \underbrace{\mathbf{A} + \ldots + \mathbf{A}}_{t} \quad \text{and} \quad \mathbf{A}^t = \underbrace{\mathbf{A} \cdot \ldots \cdot \mathbf{A}}_{t}.$$

In addition, we define one unitary operation $-$ as

$$-\mathbf{A} = \{-a\colon a \in \mathbf{A}\} \cup \{0, 1\}.$$

Observation 1.2. *Let* $\mathbf{A}, \mathbf{B} \subseteq \mathbb{R}$ *be two sets such that* $0, 1 \in \mathbf{A} \cap \mathbf{B}$. *Then*

$$\mathbf{A} \subseteq \mathbf{A} + \mathbf{B} \ \text{ and } \ \mathbf{A} \subseteq \mathbf{A} \cdot \mathbf{B}.$$

Consequently, for $s, t \in \mathbb{N}$ *such that* $s \leq t$ *we have the following two inclusions*

$$s\mathbf{A} \subseteq t\mathbf{A} \ \text{ and } \ \mathbf{A}^s \subseteq \mathbf{A}^t.$$

It the next lemma we introduce some basic properties of these operations. Although many other observations can be made, we restrict ourselves only to those that will be useful in future chapters.

Lemma 1.3. *Let* $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ *be subsets of* $\mathbb{R}$ *and* t, s, p *be natural numbers. Then the following four equations hold*

1. $\mathbf{A} \cdot (\mathbf{B} + \mathbf{C}) = \mathbf{A} \cdot \mathbf{B} + \mathbf{A} \cdot \mathbf{C}$,
2. $t\mathbf{A} \cdot s\mathbf{B} = p\mathbf{C}$, *where* $\mathbf{C} = \mathbf{A} \cdot \mathbf{B} \subseteq \mathbb{R}$ *and* $p = t \cdot s \in \mathbb{N}$,
3. $(t\mathbf{A})^s = t^s \mathbf{A}^s$,
4. $(\mathbf{A} \cup \mathbf{B})^2 = \mathbf{A}^2 \cup 2(\mathbf{A} \cdot \mathbf{B}) \cup \mathbf{B}^2$.

Proof. The lemma follows directly from Definition 1.1. $\square$

In this work we often refer to the minimal or the maximal size of an element in a set. Therefore, we establish the notation in the next definition.

Definition 1.4. Let $\mathbf{A}$ be a finite subset of $\mathbb{R}$. Then the minimal and the maximal size of an element in $\mathbf{A}$ are defined as

$$\min\nolimits_{\mathbf{A}} = \min\{|a| \colon a \in \mathbf{A} \setminus \{0\}\}, \ \ \max\nolimits_{\mathbf{A}} = \max\{|a| \colon a \in \mathbf{A}\}.$$

In the rest of this section we consider only finite subsets of $\mathbb{R}$. Let us now make a few important observations. Because we assume that $1 \in \mathbf{A}$ for every $\mathbf{A} \subseteq \mathbb{R}$, we can observe the following.

Observation 1.5. *Let* $\mathbf{A} \subseteq \mathbb{R}$. *Then both* $\min_{\mathbf{A}}, \max_{\mathbf{A}}$ *are defined and*

$$\min\nolimits_{\mathbf{A}} \leq 1, \ \ \max\nolimits_{\mathbf{A}} \geq 1.$$

In Definition 1.4 we are interested in the minimal (respectively, the maximal) element in absolute value. Therefore, we can state the next observation.

Observation 1.6. *Let* $\mathbf{A} \subseteq \mathbb{R}$. *Then*

$$\min\nolimits_{\mathbf{A}} = \min\nolimits_{-\mathbf{A}} \ \text{ and } \ \max\nolimits_{\mathbf{A}} = \max\nolimits_{-\mathbf{A}}.$$

In addition, it will be useful to express the maximal and the minimal size of an element of the sets $t\mathbf{A}, \mathbf{A}^s$ in terms of the minimal and the maximal size of an element in the set $\mathbf{A}$. Note that for $e \in \mathbb{R}$ we have

$$|e^s| \leq |e|, \ \text{ if } |e| < 1 \text{ and}$$
$$|e^s| \geq |e|, \ \text{ if } |e| \geq 1.$$

Observation 1.7. *Let* $\mathbf{A} \subseteq \mathbb{R}$ *and* $s, t \in \mathbb{N}$. *Then*

$$\min_{\mathbf{A}^s} = (\min_{\mathbf{A}})^s, \;\; \max_{\mathbf{A}^s} = (\max_{\mathbf{A}})^s \quad \textit{and} \quad \max_{t\mathbf{A}} = t \cdot \max_{\mathbf{A}}.$$

We have a reason to believe that the value $\min_{t\mathbf{A}}$ is difficult to express for a general set $\mathbf{A}$. Notice that it is of a similar nature as several famous problems which are considered to be hard, for instance, the Shortest Vector Problem (that is, the problem of finding find the shortest vector of a lattice given a basis of the lattice). Nevertheless, the value $\min_{t\mathbf{A}}$ can be very easy to express for a concrete set $\mathbf{A}$. For example, if $\mathbf{A} \subseteq \mathbb{R}^{\geq 0}$ then $\min_{t\mathbf{A}} = \min_{\mathbf{A}}$.

For a set $\mathbf{A} \subseteq \mathbb{R}$ and a natural number t we define a new function μ as

$$\mu(t, \mathbf{A}) = \min_{t\mathbf{A}}.$$

The only purpose of this new definition is the clarity of notation.

1.2.2 Semi-algebraic sets

A semi-algebraic set over an ordered ring R is a set of elements which satisfy a finite system of equations and inequalities over R. There appear many equivalent definitions of a semi-algebraic set in the literature. The definition introduced below corresponds to the one presented in the book [1].

Definition 1.8 ([1, Chapter 2])**.** Let R be an ordered commutative ring with unit and $n \in \mathbb{N}$. A set $\mathbf{S} \subset R^n$ is a *basic semi-algebraic set over* R if

$$\mathbf{S} = \{x \in R^n \colon f_1(x) < 0, \ldots, f_m(x) < 0, g_1(x) \geq 0, \ldots, g_\ell(x) \geq 0\},$$

were f_i, g_i are polynomials of n variables over R and $m, \ell \in \mathbb{N}$.

A *semi-algebraic* set is a finite union of basic semi-algebraic sets.

Semi-algebraic sets are generally very well studied and understood. They are crucial for studying o-minimal structures. The theory of o-minimality is developed in detail for example in the book [13].

Semi-algebraic sets have many nice properties, some of which we state in the next lemma. Proofs and more general variants of the statements can be found in the book mentioned above.

Lemma 1.9. *Let R be an ordered commutative ring with unit and $n \in \mathbb{N}$.*

1. *Let $\mathbf{S} \subset R^{n+1}$ be a semi-algebraic set. Then $\pi(\mathbf{S}) \subset R^n$ is a semi-algebraic set, where $\pi \colon R^{n+1} \to R^n$ is the projection map on the first n coordinates.*
2. *The intersection of two semi-algebraic sets over R is a semi-algebraic set over R.*

Since we are mainly interested in sets over real numbers, let us make the following remark.

Remark 1.10. Every semi-algebraic subset of $\mathbb{R}$ is a finite union of open intervals and points.

1.3 Polynomials and rational functions over $\mathbb{R}$

In this section we recall basic definitions and properties of polynomials and rational functions over $\mathbb{R}$. More details can be found in every textbook of algebra, for example, in the book [6].

1.3.1 Polynomials and rational functions of n variables

In this work we understand a non-zero *polynomial* $p \in \mathbb{R}[x_1, \dots, x_n]$ *of degree* $d \in \mathbb{N} \cup \{0\}$ as a sum of *monomials*

$$a_{i_1,\dots,i_n} \cdot x_1^{i_1} \cdot \ldots \cdot x_n^{i_n},$$

where $i_1, \dots, i_n \in \mathbb{N} \cup \{0\}$ are such that $\sum_{k=1}^{n} i_k \leq d$ and $a_{i_1,\dots,i_n} \in \mathbb{R}$. The expression

$$x_1^{i_1} \cdot \ldots \cdot x_n^{i_n},$$

is called a *term* in variables $x_1, \dots, x_n$. By ordering of terms in variables $x_1, \dots, x_n$ we mean the *lexicographic ordering*, i.e. the inequality

$$x_1^{i_1} \cdot \ldots \cdot x_n^{i_n} < x_1^{j_1} \cdot \ldots \cdot x_n^{j_n},$$

holds if and only if there exists a natural number $k \in \{1, \dots n\}$ such that for all $\ell < k$

$$i_\ell = j_\ell \quad \text{but} \quad i_k < j_k.$$

The *leading term* of a polynomial is the maximal term with a non-zero coefficient of the polynomial according to the ordering defined above.

The real values $a_{i_1,\dots,i_n}$ are called *coefficients* of the polynomial. The set of all coefficients of a polynomial p is denoted by $\mathbf{coef}(p)$. For technical reasons we always assume that $0, 1 \in \mathbf{coef}(p)$. The coefficient by the leading term of a polynomial p is called the *leading coefficient* of the polynomial. We denote it $\mathrm{lc}(p)$. For the degree of a polynomial p we use the standard notation which is $\deg(p)$.

For completeness, let us define the introduced terms for the zero polynomial as well. The leading coefficient of the zero polynomial is defined as $\mathrm{lc}(0) = 0$. The set of coefficients of the zero polynomial is the set $\mathbf{coef}(0) = \{0, 1\}$. And the degree of the zero polynomial is defined as $\deg(0) = 0$.

Remark 1.11. The symbol

$$p(x_1, \dots, x_n)$$

can be understood in two different ways. Either it refers to a polynomial p of variables $x_1, \dots, x_n$ or it can be the evaluation of the polynomial function p on the vector $(x_1, \dots, x_n) \in \mathbb{R}^n$. Although it might cause some confusion, we do not strictly distinguish these two cases in this work. We believe that the intended meaning will be clear from the context.

A *rational function* f of n variables over $\mathbb{R}$ is a function which can be written as a fraction of two polynomials $p, q \in \mathbb{R}[x_1, \dots, x_n]$, where q is a non-zero polynomial. The domain of the rational function f is

$$\mathrm{Dom}(f) = \mathbb{R}^n \setminus \{x \in \mathbb{R}^n \colon q(x) \neq 0\}.$$

While rational functions are elements of a particularly defined field, in computations of machines they are always represented by some (fixed) pair of polynomials as their quotient.

We define polynomial and rational maps in the standard way. Let $p_1, \ldots, p_m$ be polynomials of n variables over $\mathbb{R}$. Then the map p defined by polynomials $p_1, \ldots, p_m$ is the map

$$p\colon \mathbb{R}^n \to \mathbb{R}^m,$$
$$p((x_1, \ldots, x_n)) = (p_1((x_1, \ldots, x_n)), \ldots, p_m((x_1, \ldots, x_n))).$$

Rational maps are defined similarly. Let $f_1, \ldots, f_m$ be rational functions of n variables over $\mathbb{R}$. Then the map f defined by rational functions $f_1, \ldots, f_m$ is the map

$$f\colon \mathbb{R}^n \to \mathbb{R}^m,$$
$$f((x_1, \ldots, x_n)) = (f_1((x_1, \ldots, x_n)), \ldots, f_m((x_1, \ldots, x_n))),$$

whose domain is $\mathrm{Dom}(f) = \bigcap\limits_{i \in \{1,\ldots,m\}} \mathrm{Dom}(f_i)$.

1.3.2 Polynomials and rational functions of one variable

In this subsection we study polynomials and rational functions of one variable over $\mathbb{R}$. We state technical lemmas which will be useful in later chapters.

The first lemma describes what we know about the set of coefficients and the degree of a polynomial which is a sum (respectively, a product) of two polynomials.

Lemma 1.12. *Let $p, q \in \mathbb{R}[x]$ be two polynomials and let us define an auxiliary value*
$$d \coloneqq 1 + \min\{\deg(p), \deg(q)\}.$$
Assume that the set $\mathbf{A} \subseteq \mathbb{R}$ is such that $\mathbf{coef}(p), \mathbf{coef}(q) \subseteq \mathbf{A}$. Then

1. $\mathbf{coef}(p+q) \subseteq 2\mathbf{A}$ *and* $\deg(p+q) \leq \max\{\deg(p), \deg(q)\}$,
2. $\mathbf{coef}(p \cdot q) \subseteq d\mathbf{A}^2$ *and* $\deg(p \cdot q) \leq \deg(p) + \deg(q)$.

Proof. We have two real polynomials

$$p(x) = \sum_{i=0}^{d_p} p_i x^i, \quad q(x) = \sum_{i=0}^{d_q} q_i x^i,$$

where $p_i, q_i \in \mathbb{R}$ and $p_{d_p}, q_{d_q} \neq 0$. Without lost of generality we can assume that $d_p \leq d_q$.

1. Let us first define a new polynomial $r \in \mathbb{R}[x]$ as

$$r(x) = \sum_{i=0}^{d_r} r_i x^i = p(x) + q(x).$$

The sum can be expanded as

$$p(x) + q(x) = \sum_{i=0}^{d_p} (p_i + q_i)x^i + \sum_{i=d_p+1}^{d_q} q_i x^i.$$

From here we see that

$$r_i = \begin{cases} p_i + q_i, & \text{for } 0 \le i \le d_p, \\ q_i, & \text{for } d_p < i \le d_q. \end{cases} \tag{1.1}$$

Thus, the statement about coefficients follows from Definition 1.1 and Observation 1.2.

In addition, from the equation (1.1) we know that $d_r \le d_q$. Note that if $\deg(p) = \deg(q)$ and $\mathrm{lc}(p) + \mathrm{lc}(q) = 0$ then we have $d_r < d_q$. Otherwise the degree of the polynomial r equals the degree of the polynomial q.

2. To prove the second part of the statement, let us define a polynomial $r \in \mathbb{R}[x]$ as

$$r(x) = \sum_{i=0}^{d_r} r_i x^i = p(x) \cdot q(x).$$

We first discuss the special case when p or q is the zero polynomial. Then their product is also the zero polynomial. Thus, the degree is $\deg(r) = 0$ and the set of coefficients is $\mathbf{coef}(r) = \{0, 1\}$ by our definition. We can conclude that the statement is true in this special case.

Assume now that both p and q are non-zero polynomials. Their product can be expanded as

$$p(x) \cdot q(x) = \sum_{k=0}^{d_p+d_q} \left(\sum_{i=\max\{0,k-d_q\}}^{\min\{k,d_p\}} p_i \cdot q_{k-i} \right) x^k.$$

Hence, we derived that $d_r = d_p + d_q$. Since $\mathbf{coef}(p), \mathbf{coef}(q) \subseteq \mathbf{A}$, we have

$$p_i \cdot q_{k-i} \in \mathbf{A}^2,$$

for every pair of indices (k, i). Consequently, every coefficient of the polynomial r is a sum of at most $d_p + 1$ elements of the set $\mathbf{A}^2$.

□

We can naturally extend the previous lemma from polynomials to fractions of two polynomials; hence, to rational functions.

Lemma 1.13. *Let $p_1, q_1, p_2, q_2 \in \mathbb{R}[x]$ be polynomials such that their coefficients are elements of the set $\mathbf{A} \subseteq \mathbb{R}$ and q_1, q_2 are non-zero polynomials. Let us define two auxiliary values*

$$\begin{aligned} d_{max} &\coloneqq \max\{\deg(p_1), \deg(q_1), \deg(p_2), \deg(q_2)\}, \\ d &\coloneqq 2\,(1 + d_{max})\,. \end{aligned}$$

1. *There exist polynomials $p, q \in \mathbb{R}[x]$ such that*
$$\frac{p}{q} = \frac{p_1}{q_1} + \frac{p_2}{q_2},$$
$$\mathbf{coef}(p) \subseteq d\mathbf{A}^2,\ \mathbf{coef}(q) \subseteq \frac{d}{2}\mathbf{A}^2,$$
$$\deg(p), \deg(q) \leq 2d_{max}.$$

2. *There exist polynomials $p, q \in \mathbb{R}[x]$ such that*
$$\frac{p}{q} = \frac{p_1}{q_1} \cdot \frac{p_2}{q_2},$$
$$\mathbf{coef}(p), \mathbf{coef}(q) \subseteq \frac{d}{2}\mathbf{A}^2,$$
$$\deg(p), \deg(q) \leq 2d_{max}.$$

Proof. 1. We can express the sum of two rational functions as
$$\frac{p_1}{q_1} + \frac{p_2}{q_2} = \frac{p_1 \cdot q_2 + p_2 \cdot q_1}{q_1 \cdot q_2}.$$
From the second part of Lemma 1.12 we know that
$$\mathbf{coef}(p_1 \cdot q_2), \mathbf{coef}(p_2 \cdot q_1), \mathbf{coef}(q_1 \cdot q_2) \subseteq \frac{d}{2}\mathbf{A}^2.$$
From the first part of Lemma 1.12 it follows that
$$\mathbf{coef}(p_1 \cdot q_2 + p_2 \cdot q_1) \subseteq d\mathbf{A}^2.$$
Thus, for $p = p_1 \cdot q_2 + p_2 \cdot q_1$ and $q = q_1 \cdot q_2$ we have
$$\mathbf{coef}(q) \subseteq \frac{d}{2}\mathbf{A}^2 \text{ and } \mathbf{coef}(p) \subseteq d\mathbf{A}^2,$$
which is exactly what we wanted to prove. The statement about the degrees follows directly from Lemma 1.12.

2. Analogously, we can prove that the polynomials p and q defined as
$$p = p_1 \cdot p_2 \text{ and } q = q_1 \cdot q_2,$$
satisfy the second part of the statement.

□

In the following lemma we study the composition of two rational functions.

Lemma 1.14. *Let $p, q, r, s \in \mathbb{R}[x]$ be polynomials such that q and s are non-zero polynomials. Then the composition*
$$\frac{p}{q} \circ \frac{r}{s}$$
is a fraction of two polynomials; thus, a rational function. Moreover, the numerator and denominator of the composition are polynomials of degree less or equal to
$$d_{max}^3 + 2d_{max}^2,$$
where $d_{max} = \max\{\deg(p), \deg(q), \deg(r), \deg(s)\}$.

Explicit computation yields the degree estimates stated in the lemma. We omit the technical proof.

1.3.3 Zeros of a polynomial

Now we state two well known lemmas, which are of great importance not only in algebra. The proofs can be found in every algebraic textbook and also for example in [1] or [13]. The first lemma tells us that the number of roots of a polynomial is bounded by the degree of the polynomial. The second lemma provides a bound on the size of the roots of a polynomial. The bound depends on the size of the coefficients of the polynomial. Both lemmas can be stated more generally, i.e. for complex polynomials, but we consider only the situation over $\mathbb{R}$.

Lemma 1.15 ([1, Lemma 1 in Chapter 9])**.** *Let $p \in \mathbb{R}[x]$ be a polynomial of degree $d \geq 1$. Then*

$$|\{x \in \mathbb{R} \colon p(x) = 0\}| \leq d.$$

Lemma 1.16 ([13, Lemma 1.1])**.** *Let $\alpha \in \mathbb{R}$ be a zero of the polynomial*

$$p(x) = \sum_{i=0}^{d} p_i x^i \in \mathbb{R}[x],$$

where $p_d \neq 0$ and $d \geq 1$. Then

$$|\alpha| \leq 1 + \frac{1}{|p_d|} \max\{|p_0|, \dots |p_{d-1}|\}.$$

The next corollary will be useful later in this work. It follows from Lemma 1.16 and the Intermediate Value Theorem.

Corollary 1.17. *Let p be the polynomial from Lemma 1.16 and let*

$$L = 1 + \frac{1}{|p_d|} \max\{|p_0|, \dots |p_{d-1}|\}$$

be the bound on the size of zeros of the polynomial p. Then either

$$\begin{aligned} \forall x > L \colon\ & p(x) < 0 \ \textit{or} \\ \forall x > L \colon\ & p(x) > 0. \end{aligned}$$

Similarly, one of the following options is true. Either

$$\begin{aligned} &\forall x < -L \colon p(x) < 0 \ \textit{or} \\ &\forall x < -L \colon p(x) > 0. \end{aligned}$$

Let us end this section with a lemma in which we discuss when a rational function of one variable over $\mathbb{R}$ is a constant function. To prove the statement, we use Lemma 1.15.

Lemma 1.18. *Let f be a rational function of one variable over $\mathbb{R}$. If there exists a non-degenerated interval $I \subseteq \mathrm{Dom}(f)$ and a constant $c \in \mathbb{R}$ such that*

$$\forall x \in I \colon f(x) = c,$$

then f is the constant function c on $\mathbb{R}$ which means that

$$\forall x \in \mathbb{R} \colon f(x) = c.$$

Proof. Let p, q be two polynomials such that $f = \frac{p}{q}$. First note that for every $x \in \mathrm{Dom}(f)$ we have

$$\frac{p(x)}{q(x)} = c \Leftrightarrow p(x) - c \cdot q(x) = 0.$$

Let $I \subseteq \mathrm{Dom}(f)$ be an open interval and $c \in \mathbb{R}$ be a real constant. We now define a polynomial function

$$\psi(x) = p(x) - c \cdot q(x).$$

Since $\mathrm{Dom}(f) = \mathrm{Dom}(\psi)$, we have

$$f(x) = c \Leftrightarrow \psi(x) = 0.$$

Hence, by our assumption

$$\forall x \in I \colon \psi(x) = 0.$$

Because I is a non-degenerated interval, i.e. it is an uncountable subset of $\mathbb{R}$, the polynomial function ψ must be a constant function. Otherwise we would yield a contradiction with Lemma 1.15. Thus, we derived that ψ is the zero polynomial and consequently, the function f is the constant function c. □

2. The BSS model of computation

The BSS model of computation was presented by Lenore Blum, Michael Shub and Steve Smale in [2]. In this model of computation a real number is considered to be a unit size object which can be stored with infinite precision. Moreover, basic arithmetic operations with real numbers are done in a unit of time. We call this model *the BSS model of computation*, where BSS stands for the surnames of the authors, that is, for **B**lum, **S**hub and **S**male.

Analogously to the Turing model of computation, the main pillar of the BSS model is *a BSS machine.* Many concepts of the classical Turing model of computation can be extended to the BSS model in a natural way. In the next section we will for example define the halting set of a BSS machine or the halting time of a BSS machine. Moreover, just as in the classical model of computation, we can define complexity classes in the BSS model and study their relationships.

The BSS model is a very general model of computation as it can be defined for a general mathematical structure and not only for the real ordered field. We will discuss in Section 2.4 that the BSS model over $\mathbb{Z}_2$ is equivalent to the classical Turing model. Moreover, Turing machines can be simulated by BSS machines over any ring with unit. This is the reason why BSS machines are sometimes called "generalized Turing machines" or "real Turing machines" in case of BSS machines over $\mathbb{R}$.

The organization of this chapter is the following. Firstly, we introduce the concept of BSS machines by defining finite dimensional BSS machines over $\mathbb{R}$. These machines will be of our main interest in the rest of this work; therefore, we pay a lot of attention to them. In Section 2.2 we present an alternative point of view on finite dimensional BSS machines which allows us to analyse the number of arithmetic operations of a BSS machine more precisely. Consequently, we will be able to study the number of connected components of the halting set of a BSS machine. The is done in Section 2.3. We discuss the idea of general BSS machines in Section 2.4 and thereafter we briefly introduce the complexity theory in the BSS model of computation in Section 2.5. Furthermore, in Section 2.6 we explain what we mean by machines with "built-in constants". We close this chapter by discussing the relationship between informal algorithms and BSS machines.

Throughout this chapter we follow the approach of authors of the book [1].

2.1 Finite dimensional BSS machines

In this section we define finite dimensional BSS machines over $\mathbb{R}$ and discuss their main properties. Our aim is to introduce the concept of BSS machines on the level of formalism needed for our further work. Therefore, we omit some technical details and rather illustrate introduced notions on concrete examples. We believe that this approach provides a better understanding of BSS machines and it will not cause any confusion later in the work.

More details and formal definitions can be found in Chapter 2 of the book [1].

2.1.1 Definition

We begin with the definition of a finite dimensional machine over the real ordered field.

Definition 2.1 ([1, Definition 1 in Chapter 2])**.** A finite dimensional BSS machine M over $\mathbb{R}$ is a finite connected directed graph which has four types of nodes: input, output, branch and computation. The machine M has exactly one input node and at least one output node.

- The input node has no incoming edge and exactly one outgoing edge.
- A computation node has possibly several incoming edges and exactly one outgoing edge.
- A branch node has possibly several incoming edges and exactly two outgoing edges, one labelled "YES" and the other one with the label "NO".
- An output node has possibly several incoming edges and no outgoing edge.

In addition, the machine M has three spaces: the input space $\mathcal{I}_M = \mathbb{R}^n$, the state space $\mathcal{S}_M = \mathbb{R}^m$ and the output space $\mathcal{O}_M = \mathbb{R}^\ell$, where $n, m, \ell \in \mathbb{N}$.

Every node of the graph has an associated map of theses spaces and a successor node assignment.

- Associated with the input node is a linear map $I \colon \mathcal{I}_M \to \mathcal{S}_M$ and a unique successor node β_1.
- Every computation node η has an associated rational map $g_\eta \colon \mathcal{S}_M \to \mathcal{S}_M$ and a unique successor node β_η.
- Every branch node η has an associated polynomial function $h_\eta \colon \mathcal{S}_M \to \mathbb{R}$ which is non-zero.
  - The condition $h_\eta(x) \geq 0$ coincides with the successor node β_η^+ along the "YES" outgoing edge.
  - The condition $h_\eta(x) < 0$ coincides with the successor node β_η^- along the "NO" outgoing edge.
- Every output node η has an associated linear map $O_\eta \colon \mathcal{S}_M \to \mathcal{O}_M$ and no successor node.

In the following remark we establish the terminology, formulate several conventions and discuss technical problems which come along with the definition of a finite dimensional BSS machine.

Remark 2.2. 1. We defined finite dimensional BSS machines over the field of real numbers $\mathbb{R}$. However, BSS machines can be defined over an arbitrary commutative ring with unit R. The definition is essentially the same as Definition 2.1 except for two technical changes. Firstly, if the ring R is not a field then the computation nodes are associated with polynomial maps (not rational maps as in the case $R = \mathbb{R}$). Secondly, if R is a ring without order (for example if $R = \mathbb{C}$) then we have to modify the successor node assignment associated with a branch node. In this case, the branch condition $h_\eta(x) = 0$ is associated with the successor node β_η^+ and the condition $h_\eta(x) \neq 0$ corresponds to the successor node β_η^-.

2. We call BSS machines from Definition 2.1 "finite dimensional BSS machines" because the input space, the state and the output space are all finite dimensional. This is not the case for *general BSS machines* which we define later in this chapter; see Section 2.4 for more details.

3. In the rest of this thesis, if we say "a BSS machine", we implicitly mean a finite dimensional BSS machine over $\mathbb{R}$.

4. We index the nodes of a BSS machine by natural numbers. The finite set of all nodes of a machine M is denoted by $\mathcal{N}_M \subset \mathbb{N}$. The unique input node of a machine will always have the index 1.

5. Sometimes it is convenient to associate the identity map
$$\begin{aligned} \mathrm{id}\colon \mathcal{S}_M &\to \mathcal{S}_M, \\ x &\mapsto x, \end{aligned}$$
with every branch node. Thus, for every branch node $\eta \in \mathcal{N}_M$ we define the map g_η to be the identity map on the state space.

6. Every computation node has an associated rational map. In particular, let g be a rational map associated with some computation node. Then
$$g_\eta(x_1, \dots, x_m) = (g_1(x_1, \dots, x_m), \dots, g_m(x_1, \dots, x_m)),$$
where g_i are rational functions represented by a *fixed pair* of polynomials $p_i, q_i \in \mathbb{R}[x_1, \dots, x_m]$, where q_i is a non-zero polynomial. Recall our convention formulated in Section 1.3.

7. Every BSS machine has an associated finite set of constants, we denote it by $\mathbf{C}_M \subset \mathbb{R}$. The set consists of all coefficients of the maps associated with the nodes of the machine (this is the reason why rational functions are represented by fixed pairs of polynomials). We implicitly assume that every BSS machine has 0 and 1 as machine constants. Recall our convention from Section 1.2.

8. We have to make sure that we do not divide by zero in the computation nodes; otherwise, the machine would not be well defined. Let us observe that this technical problem can be solved by adding a new branch node in front of every computation node.

   Let η be a computation node with the associated rational map
$$g_\eta = \left(\frac{p_1}{q_1}, \dots, \frac{p_m}{q_m}\right),$$
where $p_i, q_i \in \mathbb{R}[x_1, \dots, x_m]$ are real polynomials of m variables such that for every $i \in \{1, \dots, m\}$ the polynomial q_i is non-zero. Note that
$$\left(\prod_{i=1}^{m} q_i(x_1, \dots, x_m)\right)^2 \leq 0 \;\Leftrightarrow\; \exists i \in \{1, \dots, m\}\colon\; q_i(x_1, \dots, x_m) = 0.$$

Therefore, we add a new branch node which has the associated function

$$h(x_1, \dots, x_m) = -\left(\prod_{i=1}^{m} q_i(x_1, \dots, x_m)\right)^2 .$$

It is a non-zero polynomial function since it is a product of non-zero polynomial functions. The successor node β^+ is an output node which returns some error message. The other successor node β^- is the computation node η. In the rest of this work we implicitly assume that every computation node is well defined.

Let us demonstrate the definition of a finite dimensional BSS machine on one concrete example.

Example 2.3. Let M be the BSS machine defined by the graph in Figure 2.1. The three spaces of the machine M are

$$\mathcal{I}_M = \mathbb{R},\ \ \mathcal{S}_M = \mathbb{R}^2,\ \ \mathcal{O}_M = \mathbb{R}.$$

We see that the graph of the BSS machine M consists of four nodes. Thus, the set of nodes is

$$\mathcal{N}_M = \{1, 2, 3, 4\}.$$

For every node $\eta \in \mathcal{N}_M$ we discuss in detail the associated map and the successor node assignment.

- We see that the input node $\eta = 1$ is associated with the linear map
$$\begin{aligned} I\colon \mathbb{R} &\to \mathbb{R}^2, \\ x &\mapsto (x, 0). \end{aligned}$$
The successor node is $\beta_1 = 2$.

- The node $\eta = 2$ is a branch node. The successor node coinciding with the "YES" outgoing edge is the node number 3; thus, $\beta_2^+ = 3$. The successor node along the "NO" outgoing edge is the node number 4 and therefore $\beta_2^- = 4$. The associated function is
$$\begin{aligned} h_2\colon \mathbb{R}^2 &\to \mathbb{R}, \\ (x, k) &\mapsto x - 1. \end{aligned}$$
We see that
$$h_2(x, k) \geq 0 \ \Leftrightarrow\ x \geq 1.$$
It follows that the condition $h_2(x, k) \geq 0$ coincides with the "YES" outgoing edge as required.

- The node $\eta = 3$ is a computation node and its associated map is
$$\begin{aligned} g_3\colon \mathbb{R}^2 &\to \mathbb{R}^2, \\ (x, k) &\mapsto (x - 1, k + 1). \end{aligned}$$
The successor node is in this case $\beta_3 = 2$.

- And finally the node $\eta = 4$ is an output node whose associated map is
$$\begin{aligned} O_4\colon \mathbb{R}^2 &\to \mathbb{R}, \\ (x, k) &\mapsto k. \end{aligned}$$

The set of constants of the machine M is $\mathbf{C}_M = \{-1, 0, 1\}$.

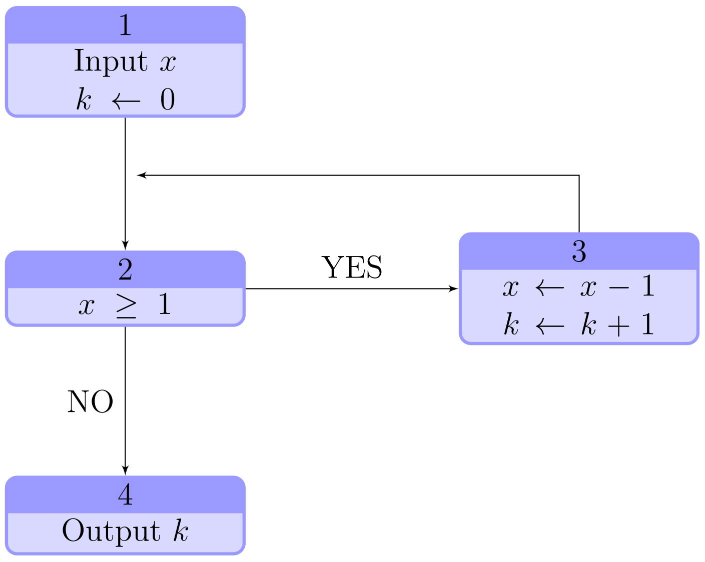


Figure 2.1: The graph of a BSS machine computing the floor function for non-negative real numbers

2.1.2 Computation of a BSS machine

We defined a BSS machine as an oriented graph whose nodes have associated linear, polynomial or rational maps and successor node assignments. In other words, every node of the graph is associated with a pair: (map, successor node assignment). This motivated the definition of computation of a BSS machine. Since the formal definition is very technical, we restrict ourselves to a heuristic definition and we illustrate it on an example. We believe that this approach does not cause any confusion and provides a better understanding of the notion. The formal definition can be found in the book [1].

Informally, given an input $x \in \mathcal{I}_M$ we follow the graph of the machine step by step. In every step of the computation we are in some "current node" and in some "current state". We execute the instruction which is associated with the node and proceed to the successor node. Let us discuss this in detail.

Firstly, we accept the input of the machine in the input node $\eta_1 = 1$ and assign the state of the machine by evaluating the associated input map. Then we proceed to the next node which is uniquely determined by the successor node assignment of the input node. Thus, the "current state" and "current node" of the machine after the first step are

$$s_2 = I(x) \text{ and } \eta_2 = \beta_1.$$

Suppose now that after $i-1$ steps we are in the state s_i and the node η_i for some $i > 1$.

If η_i is a branch node then we do not change the state of the machine. Equivalently, we can say that we apply the identity map $g_{\eta_i} = \mathrm{id}$ on the state of the machine (recall Remark 2.2). Then we evaluate the branch function and, depending on the result, we decide which outgoing edge we should follow. The "current state" and "current node" after the i-th step of the computation are

$$s_{i+1} = g_{\eta_i}(s_i) \text{ and } \eta_{i+1} = \begin{cases} \beta^+_{\eta_i}, & \text{if } h_{\eta_i}(s_i) \geq 0, \\ \beta^-_{\eta_i}, & \text{if } h_{\eta_i}(s_i) < 0. \end{cases}$$

If the node η_i is a computation node then we reassign the state of the machine by evaluating the associated computation map g_{η_i}. Then we move to the unique successor node. Thus,

$$s_{i+1} = g_{\eta_i}(s_i) \text{ and } \eta_{i+1} = \beta_{\eta_i}.$$

Finally, if the node η_i is an output node then we output the result which is computed according to the output map associated with the output node. Then the computation halts. This means that the result of the computation is $O_{\eta_i}(s_{\eta_i})$. Note that if we never enter an output node then the computation does not halt.

It follows from our description that the computation of a machine M on an input x can be described by a sequence of pairs (s_i, η_i), where $s_i \in \mathcal{S}_M$ and $\eta_i \in \mathcal{N}_M$.

It is very useful to observe that the "current state" s_i can viewed as a composition of maps; that is,

$$s_i(x) = (g_{\eta_{i-1}} \circ \dots g_{\eta_2} \circ I)(x).$$

Note that all maps in the composition are rational maps (the identity map and polynomial maps can be understood as rational maps whose denominator is 1). And composition of rational maps is a rational map (see Lemma 1.14).

Another important observation is that BSS machines (as we defined them in Definition 2.1) are deterministic machines. In other words, every input has a unique computation sequence.

Let us now perform a computation of the machine M which was defined in Example 2.3 on a concrete input.

Example 2.4. Let M be the machine defined in Example 2.3. The step by step computation of the machine M on the input π is illustrated in Figure 2.2.

2.1.3 Computation paths

By a *computation path γ of a machine M* we understand a sequence of nodes

$$\gamma = (\eta_1, \eta_2, \dots),$$

where $\eta_i \in \mathcal{N}_M$ and the first node of the sequence η_1 is the unique input node of the machine M (i.e. $\eta_1 = 1$ according to our convention). Moreover, for every other node η_i occurring in the sequence γ, one of the following is true:

$$\eta_i = \begin{cases} \beta_{\eta_{i-1}}, & \text{if } \eta_{i-1} \text{ is an input or a computation node}, \\ \beta^+_{\eta_{i-1}}, & \text{if } \eta_{i-1} \text{ is a branch node}, \\ \beta^-_{\eta_{i-1}}, & \text{if } \eta_{i-1} \text{ is a branch node}. \end{cases}$$

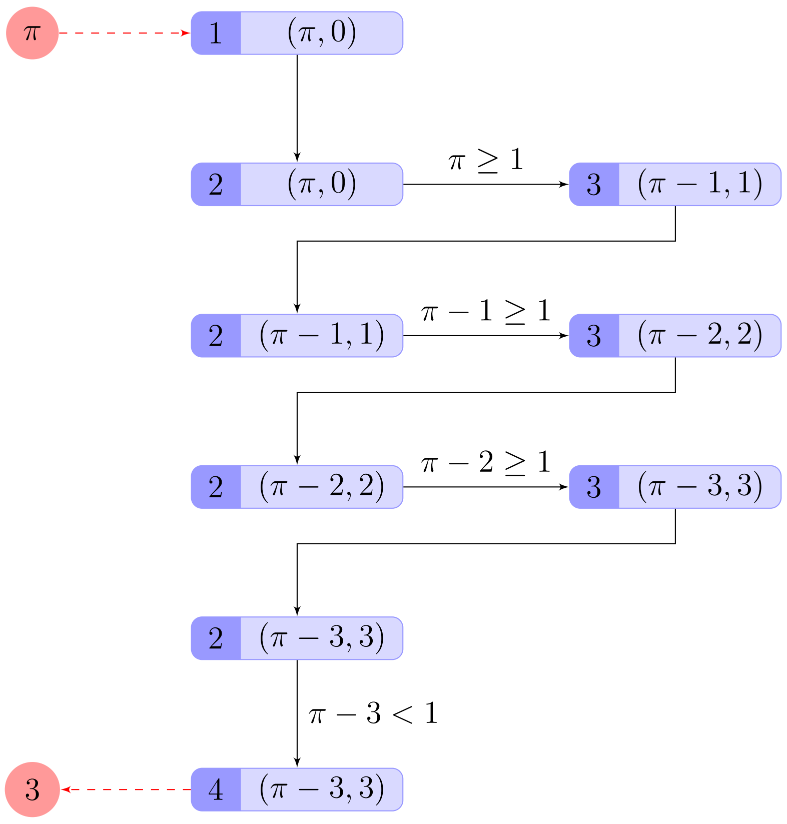


Figure 2.2: The computation of the machine M defined in Figure 2.1 on the input value π

Since an output node has no successor node, it is clear that if a sequence has an output node then it must be its last node; hence, the sequence is finite. A computation path γ of a BSS machine M is a *halting path* of the machine M if it is a finite sequence of nodes whose last node is an output node.

We define *length of a path* γ as the number of nodes in the sequence. If the sequence is infinite then we define its length as ∞. We write $\operatorname{len}(\gamma)$ for length of the path γ.

We denote by Γ_M the *set of all halting paths* of a machine M. In addition, for every $T \in \mathbb{N}$ we define the *set of all time-T halting paths* of a machine M as

$$\Gamma_{M,T} \coloneqq \{\gamma \in \Gamma_M \colon \operatorname{len}(\gamma) \leq T\}.$$

We now state and prove one simple but very useful lemma.

Lemma 2.5. *Let M be a BSS machine. Then for every natural number $T \in \mathbb{N}$ the number of halting paths of length at most T is bounded by 2^T. In particular,*

$$|\Gamma_{M,T}| \leq 2^T.$$

Proof. Length of a path is defined as the number of nodes in the sequence. Thus, if a path has at most T nodes then it has at most T branch nodes. Every branch node has exactly two successor nodes. It follows that the machine has at most 2^T different paths of length at most T. □

Note that every halting path has one input node and one output node. Hence, we can prove that

$$|\Gamma_{M,T}| \leq 2^{T-2},$$

which is a tighter bound. However, the bound stated in the lemma is sufficient for our purposes and it will be easier to work with in later chapters.

Corollary 2.6. *The set Γ_M is a countable set for every BSS machine M.*

Proof. By Lemma 2.5 we know that the set $\Gamma_{M,T}$ is finite for every $T \in \mathbb{N}$. Therefore, the set

$$\Gamma_M = \bigcup_{T \in \mathbb{N}} \Gamma_{M,T}$$

is a countable union of finite set; hence, a countable set. □

We continue with our example by discussing the computation paths.

Example 2.7. Let M be the machine defined in Example 2.3. Consider four different sequences:

$$\gamma_1 = (1,2,3,4), \;\; \gamma_2 = (1,2,3,2,4),$$
$$\gamma_3 = (1,2,3,2,3), \;\; \gamma_4 = (1,2,3,2,3,\dots).$$

We see that the only sequence which is not a valid computation path of the machine M is the sequence γ_1 (the successor node of the node 3 is the node 2 and not the node 4).

Lengths of the computation paths γ_2, γ_3 and γ_4 are

$$\operatorname{len}(\gamma_2) = 5, \;\; \operatorname{len}(\gamma_3) = 5 \;\text{ and }\; \operatorname{len}(\gamma_4) = \infty.$$

The set of all halting paths is

$$\Gamma_M = \{(1, 2, (3, 2)^n, 4) \colon n \in \mathbb{N} \cup \{0\}\},$$

where $(3, 2)^n$ denotes the concatenation of n sequences of the form $(3, 2)$. We see that only the path γ_2 is a halting path (the last node of the path γ_3 is not an output node and the length of the path γ_4 is infinite).

The set of all halting paths of length at most 6 is

$$\Gamma_{M,6} = \{(1, 2, 4), (1, 2, 3, 2, 4)\}.$$

Thus, the path γ_2 lies also in the set $\Gamma_{M,6}$.

At the end of the previous subsection we observed that every input x of a BSS machine M determines a unique computation sequence; in particular, a sequence of the form

$$((s_2, \eta_2), (s_3, \eta_3), \dots),$$

where $(s_i, \eta_i) \in \mathcal{S}_M \times \mathcal{N}_M$. We now consider the sequence of the second components of the computation sequence, namely, the sequence of nodes $(\eta_2, \eta_3 \dots)$, and define a new sequence γ_x as

$$\gamma_x = (1, \eta_2, \eta_3, \dots).$$

It is easy to verify that γ_x is a computation path of the machine M (recall that the input node is always the first node of a computation path and by our convention we label it by the index 1). We call the path γ_x *the computation path traversed by the input x*.

Note that for every input $x \in \mathcal{I}_M$, the path γ_x is either a halting path of the machine M or its length is infinite. It is also important to mention that the path traversed by an input x can be equal to the path traversed by a different input x'. Let us demonstrate it on our example.

Example 2.8. Let M be the machine defined in Example 2.3. Then the computation path traversed by the input π is

$$\gamma_\pi = (1, 2, 3, 2, 3, 2, 3, 2, 4),$$

as we can see in Figure 2.2. It is easy to observe that the input $x = \frac{16}{5}$ traverse the same path as the input π, i.e. that $\gamma_x = \gamma_\pi$.

2.1.4 The halting set and computation time

By *computation time of a machine M* we understand the function T_M which is defined as

$$\begin{aligned} T_M \colon \mathcal{I} &\to \mathbb{N} \cup \{\infty\}, \\ x &\mapsto \operatorname{len}(\gamma_x), \end{aligned}$$

where γ_x is the computation path traversed by the input x. We say that a BSS machine M *halts* on an input x if

$$T_M(x) < \infty.$$

A machine M does not halt on an input x if

$$T_M(x) = \infty.$$

Note that a machine M halts on an input x if and only if the path γ_x is a halting path so the terminology is not confusing.

We define the *halting set* of a BSS machine M in a natural way as the set

$$\Omega_M = \{x \in \mathcal{I}_M \colon T_M(x) < \infty\}.$$

In addition, for every $T \in \mathbb{N}$ we define the *time-T halting set* of the machine M as

$$\Omega_{M,T} = \{x \in \mathcal{I}_M \colon T_M(x) \leq T\}.$$

We say that the BSS machine M is *a constant time BSS machine* if there exists a natural number $T \in \mathbb{N}$ such that

$$\forall x \in \Omega_M \colon T_M(x) \leq T.$$

Or equivalently, the machine M is a constant time BSS machine if $\Omega_M = \Omega_{M,T}$ for some $T \in \mathbb{N}$.

Moreover, for every halting path $\gamma \in \Gamma_M$ we define the set $\nu_\gamma \subseteq \Omega_M$ of all inputs that traverse the path γ. In particular,

$$\nu_\gamma = \{x \in \Omega_M \colon \gamma = \gamma_x\}.$$

Let us state and prove the following useful lemma.

Lemma 2.9 ([1, Lemma 1 in Section 2.2])**.** *For every $T \in \mathbb{N}$ the time-T halting set $\Omega_{M,T}$ is a finite disjoint union of the sets ν_γ, where $\gamma \in \Gamma_{M,T}$. In particular,*

$$\Omega_{M,T} = \dot{\bigcup_{\gamma \in \Gamma_{M,T}}} \nu_\gamma.$$

The halting set Ω_M is a countable disjoint union of the sets ν_γ, where $\gamma \in \Gamma_M$. This means that

$$\Omega_M = \dot{\bigcup_{\gamma \in \Gamma_M}} \nu_\gamma.$$

Proof. The sets ν_γ are disjoint since every input x traverses exactly one computation path γ_x as we observed in Subsection 2.1.2. From Lemma 2.5 we know that the set Γ_T is finite for every $T \in \mathbb{N}$. This proves that

$$\dot{\bigcup_{\gamma \in \Gamma_{M,T}}} \nu_\gamma$$

is a finite disjoint union of sets. We know that the path γ_x is a halting path if and only if the machine M halts on the input x. This implies that the union must be equal to the time-T halting set $\Omega_{M,T}$.

From Corollary 2.6 we know that Γ_M is a countable set. Thus,

$$\dot{\bigcup_{\gamma \in \Gamma_M}} \nu_\gamma$$

is a countable disjoint union of sets. We can use the same argument as above to prove that the union is equal to the halting set Ω_M. $\square$

Let us now have a closer look at the sets ν_γ, for $\gamma \in \Gamma_M$. Two different inputs $x, x' \in \Omega_M$ traverse the same path if in every branch node of their computation the machine decides to take the same outgoing edge (the signs of the branch function are the same). Thus, every path γ determines a finite set of inequalities (sign conditions). One inequality for each branch node in the sequence. The set ν_γ is then the set of the inputs which satisfy all these inequalities determined by the path γ.

In fact, we can be more precise. Every set ν_γ is defined by finitely many *polynomial* inequalities which means that it is a basic semi-algebraic set (recall Definition 1.8). Let us briefly discuss why this is true.

Let $\gamma = \{\eta_1, \ldots, \eta_t\}$ be a computation path whose ℓ-th node is a branch node and the $(\ell + 1)$-st node is the successor node along the "YES" outgoing edge, i.e.

$$\eta_{\ell+1} = \beta^+_{\eta_\ell}.$$

From Subsection 2.1.2 we know that the machine state after $\ell - 1$ computation steps can be viewed as the rational map

$$s_{\eta_{\ell-1}} = (g_{\eta_{\ell-1}} \circ \cdots \circ g_{\eta_2} \circ I).$$

Associated with the branch node η_ℓ is the polynomial function h_{η_ℓ}. Thus, the inequality associated with the branch node η_ℓ is the rational inequality

$$(h_{\eta_\ell} \circ s_{\eta_{\ell-1}})(x) \geq 0.$$

We can assume that the denominator of the rational function $h_{\eta_\ell} \circ s_{\eta_{\ell-1}}$ never vanishes (recall Remark 2.2). Consequently, one can prove that there is a polynomial inequality which is equivalent to this rational inequality.

This informal discussion gives an intuition why the following lemma holds. We would need to introduce more detailed description of BSS machines to present the formal proof which can be found in the book [1].

Lemma 2.10 ([1, Lemma 1 in Section 2.2])**.** *Let M be a BSS machine. Then the set ν_γ is a basic semi-algebraic set for every halting path $\gamma \in \Gamma_M$.*

Let us discuss the semi-algebraic sets of the machine from our example.

Example 2.11. Let M be the BSS machine defined in Example 2.3. We assume the path

$$\gamma = (1, 2, 3, 2, 3, 2, 3, 2, 4).$$

The inequalities determined by the path γ are

$$\begin{aligned}
(h_2 \circ I)(x) \geq 0 &\;\Leftrightarrow\; x \geq 1,\\
(h_2 \circ g_3 \circ I)(x) \geq 0 &\;\Leftrightarrow\; x - 1 \geq 1,\\
(h_2 \circ g_3 \circ g_3 \circ I)(x) \geq 0 &\;\Leftrightarrow\; x - 2 \geq 1,\\
(h_2 \circ g_3 \circ g_3 \circ g_3 \circ I)(x) < 0 &\;\Leftrightarrow\; x - 3 < 1.
\end{aligned}$$

Hence, the set ν_γ is

$$\nu_\gamma = \{x \in \mathbb{R} \colon x \geq 1 \;\wedge\; x - 1 \geq 1 \;\wedge\; x - 2 \geq 1 \;\wedge x - 3 < 1\}.$$

We see that not all of the inequalities are needed in this case. In particular, we have

$$\nu_\gamma = \{x \in \mathbb{R}\colon x - 2 \geq 1 \ \wedge x - 3 < 1\} \Leftrightarrow \nu_\gamma = [3, 4).$$

Recall from Example 2.7 that every halting path is of the form $(1, 2, (3, 2)^n, 4)$ for some $n \in \mathbb{N} \cup \{0\}$. Therefore, it is not difficult to observe that

$$\nu_{(1,2,(3,2)^n,4)} = \begin{cases} \{x \in \mathbb{R}\colon x < 1\}, & \text{for } n = 0; \\ \{x \in \mathbb{R}\colon n \leq x < n + 1\}, & \text{for } n \in \mathbb{N}. \end{cases}$$

The halting set of the machine M is $\Omega_M = \mathbb{R}$.

2.1.5 The input-output map

Every BSS machine M has an associated *input-output map* Φ_M. It maps the input space $\mathcal{I}_M$ to the output space $\mathcal{O}_M$ in the following way.

If a machine M does not halt on an input $x \in \mathcal{I}_M$ then the value $\Phi_M(x)$ is not defined.

If a machine M halts on an input $x \in \mathcal{I}_M$ then we define $\Phi_M(x)$ as the value outputted by the machine M whose input was x. In particular, let $\gamma_x = \eta_1, \ldots, \eta_t$ be the computation path traversed by the input x. Then we define

$$\Phi_M(x) \coloneqq \left(O_{\eta_t} \circ g_{\eta_{t-1}} \circ \cdots \circ g_{\eta_2} \circ I\right)(x).$$

We see that the map Φ_M is a composition of the same maps for all inputs which traverse the same path. Thus, for every halting path $\gamma \in \Gamma_M$ we define the partial mapping

$$\varphi_\gamma \coloneqq \Phi_M \restriction_{\nu_\gamma} = \left(O_{\eta_t} \circ g_{\eta_{t-1}} \circ \cdots \circ g_{\eta_2} \circ I\right).$$

Recall from Remark 2.2 that if η_i is a branch node then we define the map g_{η_i} as the identity map on the state space. All maps $O_{\eta_t}, g_{\eta_{t-1}}, \ldots, g_{\eta_2}, I$ are rational maps; thus, their composition is a rational map (see Lemma 1.14). The next lemma follows from this discussion.

Lemma 2.12. *Let M be a BSS machine and $\gamma \in \Gamma_M$ its halting path. Then φ_γ is a rational map such that* $\mathrm{Dom}(\varphi_\gamma) \subseteq \nu_\gamma$.

Let us now analyse the input-output map of the BSS machine from our example.

Example 2.13. Let M be the BSS machine defined in Example 2.3. We already know that $\Omega_M = \mathbb{R}$. For a halting path $\gamma = (1, 2, (2, 3)^n, 4)$, where $n \in \mathbb{N} \cup \{0\}$, we have

$$\varphi_\gamma = \Phi_M \restriction_{\nu_\gamma} = O_4 \circ \underbrace{g_3 \circ \cdots \circ g_3}_{n} \circ I = n.$$

This follows from the definition of O_4, g_3 and I (see Exercise 2.3). In addition, we know that g_2 is the identity map $\mathrm{id}\colon \mathbb{R}^2 \to \mathbb{R}^2$. In other words, the value $\Phi_M(x)$ is equal to the number of times the machine M enters the computation node during the computation on the input x.

We can conclude that

$$\Phi_M(x) = \begin{cases} 0, & \text{if } -\infty < x < 0, \\ \lfloor x \rfloor, & \text{if } x \geq 0, \end{cases}$$

where $\lfloor x \rfloor$ denotes the maximal non-negative integer smaller or equal to the real number x.

In addition, we can express the computation time of the machine M as

$$T_M(x) = \begin{cases} 3, & \text{if } -\infty < x < 0, \\ 3 + 2 \cdot \lfloor x \rfloor, & \text{if } x \geq 0. \end{cases}$$

Consequently, we see that the machine M is not a constant time BSS machine. Using the asymptotic notation we can conclude that the halting time is $\mathcal{O}(x)$.

2.1.6 Computability in the BSS model

As in the classical Turing model of computation, we can introduce the notion of computability in the BSS model of computation. We say that a map

$$\psi \colon \mathbf{X} \to \mathbb{R}^\ell, \ \ \mathbf{X} \subset \mathbb{R}^n$$

is *computable over* $\mathbb{R}$ if it is an input-output map of some BSS machine M over $\mathbb{R}$. In particular, if

$$\mathbb{R}^n = \mathcal{I}_M, \ \ \mathbb{R}^\ell = \mathcal{O}_M \quad \text{and} \quad \mathbf{X} = \Omega_M,$$

and if the map ψ equals the map Φ_M on the set $\mathbf{X}$. We can also say that the machine M *computes the map* ψ.

So the BSS machine defined in Example 2.3 computes the floor function on non-negative real numbers.

As usual, a set $\mathbf{S} \subset \mathbb{R}^m$ is *decidable* if its characteristic function

$$\chi_{\mathbf{S}}(x) = \begin{cases} 1, & \text{if } x \in \mathbf{S}, \\ 0, & \text{otherwise;} \end{cases}$$

is a computable function.

2.1.7 The Path Decomposition Theorem

We are now prepared to state the main theorem of this chapter, the Path Decomposition Theorem. It summaries all the important properties of BSS machines which we have already discussed in the previous subsections (see Lemma 2.9, Lemma 2.10 and Lemma 2.12).

Theorem 2.14 ([1, Theorem 1 in Chapter 2]). *Let M be a finite dimensional BSS machine over $\mathbb{R}$. Then M has the following properties.*

1. *For every $T \in \mathbb{N}$ the time-T halting set of the machine M*

$$\Omega_{M,T} = \dot{\bigcup_{\gamma \in \Gamma_{M,T}}} \nu_\gamma$$

is a finite disjoint union of basic semi-algebraic sets.

2. The halting set of the machine M

$$\Omega_M = \dot{\bigcup_{\gamma\in\Gamma_M}} \nu_\gamma$$

is a countable disjoint union of basic semi-algebraic sets.

3. For any halting path $\gamma \in \Gamma_M$, the partial map

$$\varphi_\gamma = \Phi_M \restriction_{\nu_\gamma}$$

is a rational map.

The theorem provides a very powerful proof technique and it has some interesting corollaries. Let us briefly discuss some of them.

Corollary 2.15. *Every decidable set is a countable union of semi-algebraic sets.*

As a consequence, some famous sets such as the Mandelbrot set or most of the Julia sets are not decidable over $\mathbb{R}$ (see [1, Section 2.4] for more details).

The next corollary follows from Remark 1.10 in which we observed that every semi-algebraic subset of $\mathbb{R}$ is a finite union of open intervals and points.

Corollary 2.16. *Let M be a BSS machine with one dimensional input space $\mathcal{I}_M = \mathbb{R}$. Then the halting set Ω_M is a countable disjoint union of intervals and points. In particular,*

$$\Omega_M = \dot{\bigcup_{\gamma\in\Gamma_M}} \nu_\gamma = \dot{\bigcup_{\gamma\in\Gamma_M}} \left(\dot{\bigcup_{i\in\{1,\dots,m_\gamma\}}} I_{\gamma,i} \right),$$

where $I_{\gamma,i} \subseteq \nu_\gamma$ is an open interval or a point and m_γ denotes the number of components of the semi-algebraic set ν_γ.

If M is a constant time machine then its halting set is a finite disjoint union of open intervals and points.

We will discuss the number of connected components of a halting set more generally in Section 2.3.

We have already shown that the halting time of the BSS machine defined in Example 2.3 is $\mathcal{O}(x)$. It is not difficult to prove that the floor function is computable by a more efficient BSS machine whose halting time is $\mathcal{O}(\log_2(x))$. However, none of these machines is a constant time machine. Thus, a very natural question arises. Can a constant time BSS machine compute the floor function for every non-negative real number? In the next lemma we prove that there is a negative answer to this question. It is an example how we can use the Path Decomposition Theorem for proving a lower bound on the halting time of a BSS machine.

Lemma 2.17 ([1, Proposition 3 in Chapter 2])**.** *Let M be a BSS machine such that given an input $x \in \mathbb{R}^{\geq 0}$ outputs $\lfloor x \rfloor$, i.e. the maximal non-negative integer which is smaller or equal to x. Then the machine M is not a constant time BSS machine.*

Proof. Let M be a BSS machine from the statement. Thus, its halting set contains all non-negative real numbers, i.e. $\mathbb{R}^{\geq 0} \subseteq \Omega_M$. Moreover, by Corollary 2.16 we know that the halting set Ω_M is a countable disjoint union of open intervals and points.

Let us fix an arbitrary positive integer $k \in \mathbb{N}$ and consider the half-closed interval $[k, k+1)$. Since the interval $[k, k+1)$ is a subset of the halting set Ω_M, the interval must contain an open subinterval J such that $J \subseteq \nu_\gamma$ for some $\gamma \in \Gamma_M$. This implies that the associated rational function φ_γ is such that

$$\forall x \in J\colon \varphi_\gamma(x) = k.$$

Since J is an open subinterval of $\mathbb{R}$, the function φ_γ is the constant function k by Lemma 1.18.

It follows that we need at least K halting paths in order to compute the floor function on the interval $[0, K)$. Consequently, we need infinitely many halting paths to compute the floor function for every non-negative real number. This implies that M cannot be a constant time machine since the number of halting paths of a constant time machine is bounded (recall Lemma 2.5). $\square$

An important consequence of the lemma is that in the BSS model of computation we cannot access the digits of a real number in constant time.

2.2 Elementary instructions

We claimed at the beginning of this chapter that in the BSS model of computation we consider one arithmetic operation as a unit time operation. However, in the previous section we defined the halting time of a BSS machine M on an input x as the number of nodes on the computation path traversed by the input x. Every node of a machine is associated with a linear, polynomial or rational map. Thus, there are several arithmetic operations performed in every node.

Therefore, in this section we formally define what we mean by one elementary instruction of a BSS machine and we introduce elementary BSS machines, this is, machines performing only one arithmetic operation at a time. Then we discuss the relationship between the halting time of BSS machines and the halting time of elementary BSS machines.

In this section M denotes a finite dimensional BSS machine over $\mathbb{R}$. Let us recall that every finite dimensional machine has there spaces; the input space $\mathcal{I}_M = \mathbb{R}^n$, the state space $\mathcal{S}_M = \mathbb{R}^m$ and the output space $\mathcal{O}_M = \mathbb{R}^\ell$ where $m, n, \ell \in \mathbb{N}$. In addition, it has the set of constants $\mathcal{C}_M$. We will omit the indices and write only $\mathcal{I}, \mathcal{S}, \mathcal{O}, \mathbf{C}$ to simplify the notation.

Let $s = (s_1, \ldots, s_m) \in \mathcal{S}$ be a state of a BSS machine M. We call $s_i \in \mathbb{R}$ a *component of the state* s, where $i \in \{1, \ldots, m\}$. Usually, we will refer to a component of the state s in a general manner. If we need to be more specific, we say the i-th component of the state s. Components of an input and an output are defined analogously.

In the following definition we formalize what we mean by an elementary instruction of a BSS machine.

Definition 2.18. Let M be a BSS machine with a set of machine constants $\mathbf{C}$ such that $0, 1 \in \mathbf{C}$. Let y be a component of a machine state. By an *elementary instruction* we mean one of the following assignments

1. $y \leftarrow c$, where $c \in \mathbf{C}$ is a machine constant,
2. $y \leftarrow y_1 + y_2$, where y_1, y_2 are components of a machine state,
3. $y \leftarrow y_1 \cdot y_2$, where y_1, y_2 are components of a machine state,
4. $y \leftarrow \frac{1}{y_1}$, where $y_1 \neq 0$ is a component of a machine state,
5. $y \leftarrow -y_1$, where y_1 is a component of a machine state.

For technical reasons if $c \in \mathbf{C}$ is a constant then $\frac{1}{c}$ is considered to be a rational function defined by the fixed pair of constant polynomials 1 and c.

Intuitively, an elementary BSS machine is a BSS machine which performs arithmetic operation only in computation nodes. Moreover, every computation node coincides with exactly one elementary instruction. Let us be more precise in the following definition.

Definition 2.19. Let M be a BSS machine with the input space $\mathcal{I} = \mathbb{R}^n$, the state space $\mathcal{S} = \mathbb{R}^m$ and the output space $\mathcal{O} = \mathbb{R}^\ell$. The set of machine constants is denoted by $\mathbf{C}$ and set of nodes is denoted by $\mathcal{N}$. We call the BSS machine M *an elementary BSS machine* if the following holds.

- The map $I \colon \mathcal{I} \to \mathcal{S}$ associated with the input node is defined by linear functions $f_1, \ldots, f_m$ which are such that for every $i \in \{1, \ldots, m\}$

$$f_i((x_1, \ldots, x_n)) = \begin{cases} x_j, & \text{for some } j \in \{1, \ldots, n\} \text{ or} \\ c, & \text{for some } c \in \mathbf{C}. \end{cases}$$

- Every computation node $\eta \in \mathcal{N}$ performs exactly one elementary instruction. In particular, the associated map $g_\eta \colon \mathcal{S}_M \to \mathcal{S}_M$ is defined by rational functions $f_1, \ldots, f_m$, where exactly one of them is not of the form

$$f_i((x_1, \ldots, x_m)) = x_i.$$

  The remaining function, let us call it f_j, is such that

$$f_j((x_1, \ldots, x_m)) = \begin{cases} c, & \text{for some } c \in \mathbf{C} \text{ or} \\ x_k + x_l, & \text{for some } k, l \in \{1, \ldots, m\} \text{ or} \\ x_k \cdot x_l, & \text{for some } k, l \in \{1, \ldots, m\} \text{ or} \\ \frac{1}{x_k}, & \text{for some } k \in \{1, \ldots, m\} \text{ or} \\ -x_k, & \text{for some } k \in \{1, \ldots, m\}. \end{cases}$$

- Every branch node $\eta \in \mathcal{N}$ has an associated function $h_\eta \colon \mathcal{S}_M \to \mathbb{R}$ which is of the form

$$h_\eta((x_1, \ldots, x_m)) = x_j, \quad \text{for some } j \in \{1, \ldots, m\}.$$

- Every output node $\eta \in \mathcal{N}$ has an associated map $O_\eta \colon \mathcal{S}_M \to \mathcal{O}_M$ which is of the form

$$O_\eta((x_1, \ldots, x_\ell, x_{\ell+1}, \ldots, x_m)) = (x_1, \ldots, x_\ell).$$

It follows from the definition that computation nodes of elementary machines can be viewed as elementary instructions.

2.2.1 The number of elementary instructions

Let us now discuss that every standard BSS machine can be simulated by an elementary BSS machine. By a standard machine we mean a BSS machine as defined in Definition 2.1.

We know that every node of a standard BSS machine is associated with a map (linear, polynomial or rational). We claim that each of these maps can be expressed by a set of elementary instructions. Let us demonstrate it on an input node.

Assume that M is a standard BSS machine with the machine spaces $\mathcal{I} \subseteq \mathbb{R}^n$ and $\mathcal{S} \subseteq \mathbb{R}^m$. The input node coincides with a linear map

$$I \colon \mathcal{I} \to \mathcal{S}.$$

Since the map is linear, every component s_i of the initial state s can be expressed as

$$s_i = \sum_{i=1}^{n} a_i x_i + b,$$

where $a_1, \ldots, a_n, b \in \mathbf{C}$ and $x_1, \ldots, x_n$ are input components. We see that at most $2 \cdot n$ elementary instructions are needed to initialize one component of the machine state. All together, at most $2 \cdot m \cdot n$ elementary instructions are needed in order to compute the linear map which corresponds to the input node.

Similarly, we can argue that every computation, branch and output node can be expressed by a set of elementary instructions. We omit the details since, especially for computation nodes, the analysis becomes rather lengthy and technical.

Nevertheless, the example gives an intuition why the number of elementary instructions depends on the dimension of the input space, the state space and the output space and the degree of the maps associated with the nodes of the machine. For every finite dimensional machine M all these parameters are fixed (they do not depend on the input). Let us conclude this discussion with the following observation.

Observation 2.20. *Let M be a finite dimensional BSS machine. Then there is an elementary BSS machine M' such that*

1. *the machines M and M' has the same machine constants, i.e. $\mathbf{C}_M = \mathbf{C}_{M'}$;*
2. *the machines M and M' has the same input-output map and the same halting set, i.e.*
$$\Phi_M = \Phi_{M'} \text{ and } \Omega_M = \Omega_{M'};$$
3. *the halting times of the machines M and M' differ only by a constant factor, i.e. for every $x \in \Omega_M = \Omega_{M'}$ the following inequalities hold*
$$T_M(x) \leq T_{M'}(x) \leq \mathcal{O}(T_M(x)).$$

2.2.2 The degree and coefficients of the input-output map

We will now study the following question. How does one elementary instruction influence the degree and the coefficients of the input-output map? This technical analysis will be useful later in the work where we will study functions that can be computed by BSS machines with one dimensional input spaces. Therefore, we restrict ourselves to this special case and for the rest of this subsection we assume that M is a finite dimensional elementary BSS machine whose input space is $\mathcal{I} = \mathbb{R}$.

Recall from Subsection 2.1.2 that every component of a machine state at any time can be viewed as a rational function over $\mathbb{R}$.

Let us first discuss the degree and the coefficients of a component of a machine state before the first computation node (i.e. before the first elementary instruction).

Lemma 2.21. *Let M be an elementary BSS machine with the input space $\mathcal{I} \subseteq \mathbb{R}$, the state space $\mathcal{S} \subset \mathbb{R}^m$ and the set machine constants $\mathbf{C}$. Let s denote the state of the machine M after the input node. Let y be a component of the state s. Then $y \in \mathbb{R}[x]$ is a polynomial such that*

$$\deg(y) \leq 1 \quad \textit{and} \quad \mathbf{coef}(y) \subseteq \mathbf{C}.$$

Proof. By definition we know that either $y = x$, where x is the input of the machine, or $y = c$, where $c \in \mathbf{C}$ is a machine constant. In the first case

$$\deg(y) = 1 \text{ and } \mathbf{coef}(y) = \{0, 1\} \subseteq \mathbf{C}$$

In the case when $y = c$ we have

$$\deg(y) = 0 \leq 1 \text{ and } \mathbf{coef}(y) = \{0, 1, c\} \subseteq \mathbf{C}.$$

□

In the next lemma we study how one elementary instruction (one computation node) influences the degree and the set of coefficients of a machine state.

Lemma 2.22. *Suppose that M is an elementary BSS machine in a state $s \in \mathcal{S}$. Let $\mathbf{A} \subseteq \mathbb{R}$ and $d_{max} \in \mathbb{N}$ be such that for every component y of the state s there exist polynomials $p, q \in \mathbb{R}[x]$ such that*

$$y = \frac{p}{q},$$

$$\deg(p), \deg(q) \leq d_{max}, \tag{2.1}$$

$$\mathbf{coef}(p), \mathbf{coef}(q) \subseteq \mathbf{A}. \tag{2.2}$$

Let s' be a state of the machine M after exactly one elementary instruction. Then for every component y' of the new state s' there exist polynomials $p', q' \in \mathbb{R}[x]$ such that

$$y' = \frac{p'}{q'},$$

$$\deg(p'), \deg(q') \leq 2d_{max}, \tag{2.3}$$

$$\mathbf{coef}(p'), \mathbf{coef}(q') \subseteq d\mathbf{B}^2, \tag{2.4}$$

where $d = 2(1 + d_{max})$ and $\mathbf{B} = \mathbf{A} \cup -\mathbf{A}$.

Proof. Since y is a component of the state s, we know by Lemma 2.21 that $\mathbf{C} \subseteq \mathbf{A}$. In addition, by Lemma 1.3 we know that

$$(\mathbf{A} \cup -\mathbf{A})^2 = \mathbf{A}^2 \cup 2\mathbf{A}(-\mathbf{A}) \cup (-\mathbf{A})^2.$$

It follows that $\mathbf{A}^2 \subseteq \mathbf{B}^2$ and $-\mathbf{A} \subseteq (-\mathbf{A})^2 \subseteq \mathbf{B}^2$.

Let us now proceed by a case analysis splitting on the type of elementary instruction.

1. Let $y' = c$, where $c \in \mathbf{C}$ is a machine constant. Then the constant polynomials $p' = c$ and $q' = 1$ trivially satisfy the statement.

2. Let $y' = y_1 + y_2$, where y_1, y_2 are components of the state s. Then from the first part of Lemma 1.13 we know that there exist polynomials $p', q' \in \mathbb{R}[x]$ such that $\frac{p'}{q'} = y_1 + y_2$, the inequality (2.3) holds and

$$\mathbf{coef}(p') \subseteq d\mathbf{A}^2, \ \ \mathbf{coef}(q') \subseteq \frac{d}{2}\mathbf{A}^2,$$

   which implies that the inclusion (2.4) is true.

3. Let $y' = y_1 \cdot y_2$, where y_1, y_2 are components of the state s. Then from the second part of Lemma 1.13 we know that there exist polynomials $p', q' \in \mathbb{R}[x]$ such that $\frac{p'}{q'} = y_1 \cdot y_2$, the inequality (2.3) holds and

$$\mathbf{coef}(p'), \mathbf{coef}(q') \subseteq \frac{d}{2}\mathbf{A}^2,$$

   which proves the inclusion (2.4).

4. Let $y' = \frac{1}{y_1}$, where $y_1 \neq 0$ is a component of the state s. Then by our assumption there exist polynomials $p, q \in \mathbb{R}[x]$ such that $y_1 = \frac{p}{q}$ and the equalities (2.1) and (2.2) hold. It is easy to observe that polynomials $p' = q$ and $q' = p$ satisfy the statement. Note that p is a non-zero polynomial since $y_1 \neq 0$.

5. Let $y' = -y_1$, where y_1 is a component of the state s. Then by our assumption there exist polynomials $p, q \in \mathbb{R}[x]$ such that $y_1 = \frac{p}{q}$ and the equalities (2.1) and (2.2) hold. Observe that

$$\mathbf{coef}(-p) \subseteq -\mathbf{coef}(p) \subseteq -\mathbf{A} \subseteq \mathbf{B}$$

   holds. It follows that polynomials $p' = -p$ and $q' = q$ satisfy the statement.

□

In the next theorem we generalize the previous lemma.

Theorem 2.23. *Assume that M is an elementary BSS machine with the set of constants $\mathbf{C}$. Let k be a positive integer. Suppose that $s \in \mathcal{S}$ is a state of the machine M after $k-1$ elementary instructions. Then for every component y of the state s there exist polynomials p, q such that the following is true:*

$$\begin{aligned} &y = \frac{p}{q}, \\ &\deg(p), \deg(q) \le 2^k, \\ &\mathbf{coef}(p), \mathbf{coef}(q) \subseteq 2^{2^{k+1}} \widetilde{\mathbf{C}}^{2^k}, \end{aligned}$$

where $\widetilde{\mathbf{C}} = \mathbf{C} \cup -\mathbf{C}$.

The proof of the statement consists of many technical computations. For better orientation, we first establish some notation which will be used in the proof.

By y_k we denote a component of a state after k elementary instructions. Let $d_k \in \mathbb{N} \cup \{0\}$ be such that for every y_k there exist polynomials p_k, q_k with the following properties

$$y_k = \frac{p_k}{q_k} \quad \text{and} \quad \deg(p_k), \deg(q_k) \le d_k.$$

And finally, for $k \ge 2$ we define an auxiliary value $D_k = 2(1 + d_{k-1})$.

Let us now prove the theorem.

Proof. From Lemma 2.21 we know that before the first elementary instruction, i.e. for $k = 1$, there exist polynomials p_1, q_1 such that $y_1 = \frac{p_1}{q_1}$ and

$$\deg(p_1), \deg(q_1) \le 1 < 2^1 \quad \text{and} \quad \mathbf{coef}(p_1), \mathbf{coef}(q_1) \subseteq \mathbf{C}.$$

Thus, the theorem holds for $k = 1$.

Let us now prove an auxiliary statement. For every $k \ge 2$ there exists polynomials p_k, q_k such that $y_k = \frac{p_k}{q_k}$, $\deg(p_k), \deg(q_k) \le 2^k$ and

$$\mathbf{coef}(p_k), \mathbf{coef}(q_k) \subseteq \left(\prod_{i=0}^{k-2} (D_{k-i})^{2^i} \right) \widetilde{\mathbf{C}}^{2^k}, \tag{2.5}$$

where $\widetilde{\mathbf{C}} = \mathbf{C} \cup -\mathbf{C}$. We proceed by induction on k.

- By Lemma 2.22 we know that there exist polynomials p_2, q_2 such that

$$\begin{aligned} &y_2 = \frac{p_2}{q_2}, \\ &\deg(p_2), \deg(q_2) \le 2 \cdot 2, \\ &\mathbf{coef}(p_2), \mathbf{coef}(q_2) \subseteq D_2 \widetilde{\mathbf{C}}^2. \end{aligned}$$

- Now we assume that the auxiliary statement is true for $k-1$. Then by Lemma 2.22 there exist polynomials p_k, q_k such that

$$\begin{aligned} &y_k = \frac{p_k}{q_k}, \\ &\deg(p_k), \deg(q_k) \le 2^k, \\ &\mathbf{coef}(p_k), \mathbf{coef}(q_k) \subseteq D_k \left(\left(\prod_{i=0}^{k-3} (D_{k-i-1})^{2^i} \right) \widetilde{\mathbf{C}}^{2^{k-1}} \right)^2. \end{aligned}$$

By Lemma 1.3 we can modify the expression on the right hand side of the inclusion as follows

$$D_k\left(\left(\prod_{i=0}^{k-3}(D_{k-i-1})^{2^i}\right)\widetilde{\mathbf{C}}^{2^{k-1}}\right)^2 = D_k\left(\left(\prod_{i=0}^{k-3}(D_{k-i-1})^{2^{i+1}}\right)\widetilde{\mathbf{C}}^{2^k}\right).$$

This can be further simplified as

$$D_k\left(\left(\prod_{i=0}^{k-3}(D_{k-i-1})^{2^{i+1}}\right)\widetilde{\mathbf{C}}^{2^k}\right) = \left(\prod_{i=0}^{k-2}(D_{k-i})^{2^i}\right)\widetilde{\mathbf{C}}^{2^k}.$$

We derived that the inclusion (2.5) holds and hence proved the auxiliary statement.

To complete the proof of the theorem we need to show that for every $k \geq 2$ the following inequality holds

$$\prod_{i=0}^{k-2}(D_{k-i})^{2^i} \leq 2^{2^{k+1}}. \tag{2.6}$$

Let us fix an arbitrary value $k \geq 2$. We have already proved that $d_{k-1} \leq 2^{k-1}$. Therefore, be can bound the value D_k as follows

$$D_k = 2(1 + d_{k-1}) \leq 2(1 + 2^{k-1}) \leq 2^{k+1}.$$

This implies that

$$\prod_{i=0}^{k-2}(D_{k-i})^{2^i} \leq \prod_{i=0}^{k-2}(2^{k-i+1})^{2^i} = \prod_{i=0}^{k-2}2^{(k-i+1)\cdot 2^i} = 2^{\sum_{i=0}^{k-2}(k-i+1)\cdot 2^i}$$

holds. The sum in the exponent can be expanded as

$$\sum_{i=0}^{k-2}(k-i+1)\cdot 2^i = k\sum_{i=0}^{k-2}2^i - \sum_{i=0}^{k-2}2^i i + \sum_{i=0}^{k-2}2^i. \tag{2.7}$$

Recall the following two well known sums

$$\begin{aligned}\sum_{i=0}^{n}2^i &= 2^{n+1} - 1,\\ \sum_{i=0}^{n}2^i \cdot i &= 2\cdot(2^n n - 2^n + 1).\end{aligned}$$

Using these general formulae, we can modify the equality (2.7) as

$$\begin{aligned}\sum_{i=0}^{k-2}(k-i+1)\cdot 2^i &= k(2^{k-1}-1) - 2(2^{k-2}(k-2) - 2^{k-2} + 1) + (2^{k-1} - 1)\\ &= 2^{k-1}k - k - 2^{k-1}k + 2^k + 2^{k-1} - 2 + 2^{k-1} - 1\\ &= 2^{k+1} - k - 3\\ &\leq 2^{k+1}.\end{aligned}$$

Thus, we proved the inequality (2.6).

And finally, using this inequality in the relation (2.5) we achieve that

$$\mathbf{coef}(p_k), \mathbf{coef}(q_k) \subseteq \left(\prod_{i=0}^{k-2} (D_{k-i})^{2^i} \right) \widetilde{\mathbf{C}}^{2^k} \subseteq 2^{2^{k+1}} \widetilde{\mathbf{C}}^{2^k},$$

which is exactly what we wanted to prove. □

Let us make one important remark. All the proofs in this subsection were constructive. This means that they provide an algorithm how to find a pair of polynomials with stated properties for every component of a machine state. From now on, by the fixed pair of polynomials whose quotient represents a component of the machine state we understand the polynomials defined by this algorithm.

2.3 The number of connected components of the halting set

In this section, we study the number of connected components of the halting set of a constant time BSS machine. Recall first what we know from Theorem 2.14 (the Path Decomposition Theorem). Let M be a constant time BSS machine. Then its halting set Ω_M is a finite disjoint union of semi-algebraic sets ν_γ, where $\gamma \in \Gamma_M$.

Since the machine M is a constant time machine, there must exist a natural number T which bounds the number of nodes on a path $\gamma \in \Gamma_M$. The number of branch nodes, let us denote it B, must be less or equal to the number of all nodes; i.e. $B \leq T$. Every branch node has two successor nodes and therefore the number of paths (respectively, semi-algebraic sets) is bounded by the value 2^T. It follows that the number of connected components of the halting set Ω_M is less or equal to

$$2^T \cdot \max_{\gamma \in \Gamma_M} \{\text{the number of connected components of the set } \nu_\gamma\}.$$

Thus, it is sufficient to find an upper bound on the number of connected components of a semi-algebraic sets ν_γ.

We begin with a lemma that bounds the number of connected components of a general semi-algebraic set. We omit the proof which can be found in the book [1].

Lemma 2.24 ([1, Proposition 3 in Chapter 16]). *Let* $\mathbf{S} \subset \mathbb{R}^n$ *be a semi-algebraic set defined by*

$$\begin{gathered} f_1(x) = 0, \ldots, f_p(x) = 0, \\ f_{p+1}(x) \geq 0, \ldots, f_{p+k}(x) \geq 0, \\ f_{p+k+1}(x) > 0, \ldots, f_{p+\ell}(x) > 0, \end{gathered}$$

where f_i *are polynomials. Let* $d \coloneqq \max\{2, \deg(f_1), \ldots, \deg(f_{p+\ell})\}$. *Then the number of connected components of the semi-algebraic set* $\mathbf{S}$ *is at most*

$$d(2d-1)^{n+\ell-1}.$$

Now we are prepared to state and prove the main theorem of this section.

Theorem 2.25. *Let M be a constant time elementary BSS machine with the input space $\mathcal{I} = \mathbb{R}^n$. Let $T \in \mathbb{N}$ be the time bound of the machine M, i.e. for every $x \in \Omega_M$ it holds that $T_M(x) \leq T$. Then for every halting path $\gamma \in \Gamma_M$ the number of connected components of the semi-algebraic set ν_γ is at most*

$$2 \cdot 3^{n+T-1}.$$

The main idea of the proof is similar to the idea used in the proof of Theorem 2 presented in the book [1, Chapter 16].

Proof. Let $x \in \mathbb{R}^n$ and let $\gamma_x = (\eta_1, \ldots, \eta_t)$ be the path traversed by the input x. From our assumption we know that $t \leq T$. Let us construct a system of quadratic equations and inequalities. For every $i \in \{1, \ldots, t\}$ we do one of the following:

- if η_i is an input node or an output node then do nothing;
- if η_i is a computational node performing an elementary instruction of the first type (i.e. the instruction $y \leftarrow c$, where c is a machine constant) then do nothing;
- if η_i is a computational node performing any other elementary instruction then add a new variable y_i and the equation

$$\begin{aligned} y_i &= s_1 + s_2, \quad \text{if } \eta_i \text{ performes addition,} \\ y_i &= s_1 \cdot s_2, \quad \text{if } \eta_i \text{ performes multiplication,} \\ y_i &= -s_1, \quad \text{if } \eta_i \text{ performes negation,} \\ s_1 \cdot y_i &= 1, \quad \text{if } \eta_i \text{ performes inversion,} \end{aligned} \tag{2.8}$$

  where $s_1, s_2 \in \{x_1, \ldots, x_n, y_1, \ldots, y_{i-1}\}$;
- if η_i is a branch instruction then add a new inequality

$$\begin{aligned} s_k &\geq 0, \text{ if } \eta_{i+1} = \beta^+_{\eta_i}, \\ s_k &< 0, \text{ if } \eta_{i+1} = \beta^-_{\eta_i}, \end{aligned} \tag{2.9}$$

  where $s_k \in \{x_1, \ldots, x_n, y_1, \ldots, y_{i-1}\}$ is the value whose sign is tested.

Let h denote the number of computational nodes which do not perform an elementary instruction of the first type. It is clear that the inequalities

$$h \leq t \leq T$$

hold. Let $\mathbf{S}$ be the semi-algebraic set defined by the system of quadratic equalities (2.8) and inequalities (2.9). Note that the set of variables is

$$\{x_1, \ldots, x_n, y_1, \ldots, y_h\};$$

hence, $\mathbf{S} \subset \mathbb{R}^{n+h}$. The set $\mathbf{S}$ is a semi-algebraic set defined by polynomial inequalities and equations of degree at most 2. Therefore, by Lemma 2.24 the set $\mathbf{S}$ has at most

$$2(2 \cdot 2 - 1)^{n+h-1} \leq 2 \cdot 3^{n+T-1}$$

connected components. The semi-algebraic set ν_γ is the projection onto the first n coordinates of the set $\mathbf{S}$. The number of connected components cannot increase during the projection; therefore, we can conclude that the number of connected components of ν_γ is at most $2 \cdot 3^{n+T-1}$ as stated in the theorem. $\square$

Corollary 2.26. *Let M be a constant time elementary BSS machine with the input space $\mathcal{I} = \mathbb{R}^n$. Let $T \in \mathbb{N}$ be the time bound of the machine M, i.e. for every $x \in \Omega_M$ it holds that $T_M(x) \leq T$. Then the number of connected components of the halting set Ω_M is bounded by*

$$2^{T+1}3^{n+T-1}.$$

2.4 General BSS machines

An important result in the classical Turing model of computation is the existence of a Turing machine which is able to compute any computable function; a universal Turing machine. We claimed in the introduction that the BSS model of computation generalize the Turing model of computation. Therefore, it is natural to ask whether there is a universal BSS machine.

One of the basic properties of a universal machine is that it accepts every finite dimensional input. However, the input space of a BSS machine, as we defined it in Section 2.1, is of fixed dimension over $\mathbb{R}$. To achieve a universal BSS machine, we need to generalize our definition. Therefore, we introduce BSS machines which can store and work with finite but unbounded sequences of elements of R, where R is an arbitrary commutative ring with unit. We call these machines *general BSS machines over R.*

Later in this work we will always consider finite dimensional BSS machines. Therefore, we present only an informative introduction to the theory of general BSS machines. Formal definition and all the details can be found in the book [1, Chapter 3].

As already mentioned, we would like to work with finite but unbounded sequence of elements of R. Following the approach of authors of the book [1], we introduce two spaces R^∞ and R_∞.

The space R^∞ is defined as the disjoint union

$$R^\infty = \dot{\bigcup_{n\geq 0}} R^n,$$

where R^0 is the 0-dimensional space with just one element $\mathbf{0}$. For $x \in R^n \subset R^\infty$ we call n the *length* of x.

The R_∞ is defined as the *bi-infinite directed sum* space over $\mathbb{R}$. In particular, the elements of the space R_∞ have the form

$$x = (\dots, x_{-2}, x_{-1}, x_0 \ \textbf{.}\ x_1, x_2, x_3, \dots),$$

where $x_i \in R$ for all $i \in \mathbb{Z}$ and $x_k = 0$ for $|k|$ large enough. The symbol " **.** " is the *distinguished marker* between x_0 and x_1.

The definition of a general BSS machine over R is similar to the definition of a finite dimensional BSS machine over R (see Definition 2.1). There are two main changes. Firstly, the spaces associated with a general machine have to be infinite dimensional spaces R^∞ and R_∞. This allows a machine to input, store and output unbounded sequences. However, to access and compute with these sequences, we need introduce another type of node; a shift node. Its function is to shift the state of the machine one coordinate to the left or to the right. Let us be more precise.

A general BSS machine M is a finite connected directed graph which has five types of nodes: input, computation, branch, output and shift. The machine has three associated spaces: the input space $\mathcal{I}_M = R^\infty$, the state space $\mathcal{S}_M = R_\infty$ and the output space $\mathcal{O}_M = R^\infty$.

The first four types of nodes are defined as in the finite dimensional case except for the associated maps.

The input node is associated with a map $I\colon R^\infty \to R_\infty$ defined as

$$(x_1, \dots, x_n) \mapsto (\dots, 0, \underbrace{1, \dots, 1}_{n} \centerdot x_1, \dots, x_n, 0, \dots).$$

The output node is associated with a map $O\colon R_\infty \to R^\infty$ defined as

$$(\dots, 0, \underbrace{1, \dots, 1}_{\ell} \centerdot x_1, \dots, x_\ell, \dots) \mapsto \begin{cases} (x_1, \dots, x_\ell) \in R^\ell, & \text{if } \ell > 0; \\ \mathbf{0} \in R^0, & \text{if } \ell = 0. \end{cases}$$

A branch node η is associated with a polynomial function $h_\eta\colon R^m \to R$ for some $m \in \mathbb{N}$. The polynomial function h_η is non-zero and it defines a polynomial function $\widehat{h_\eta}\colon R_\infty \to R$ as follows

$$\widehat{h_\eta}((\dots, x_0 \centerdot x_1, \dots x_m, x_{m+1}, \dots)) = h_\eta(x_1, \dots x_m).$$

A computation node η is associated with a rational map $g_\eta\colon R^m \to R^m$ for some $m \in \mathbb{N}$. The rational map g_η defines a rational map $\widehat{g_\eta}\colon R_\infty \to R_\infty$ as follows

$$\widehat{g_\eta}((\dots, x_0 \centerdot x_1, \dots x_m, x_{m+1}, \dots)) = (\dots, x_0 \centerdot g_\eta(x_1, \dots x_m), x_{m+1}, \dots).$$

Let us now define the fifth type of node. A shift node η has possibly several incoming edges and one outgoing edge. Thus, it is associated with a unique successor node β_η. In addition, the shift node η has an associated map $g_\eta \in \{\sigma_r, \sigma_\ell\}$, where σ_r, σ_ℓ are the shift maps defined as follows. The right shift map

$$\begin{gathered} \sigma_r\colon R_\infty \to R_\infty, \\ (\dots, x_{-1}, x_0 \centerdot x_1, x_2, \dots) \mapsto (\dots, x_{-1}, x_0, x_1 \centerdot x_2, \dots), \end{gathered}$$

shifts all the elements of the sequence one coordinate the left (equivalently we could say that it shifts the distinguished mark to the right). The left shift map

$$\begin{gathered} \sigma_\ell\colon R_\infty \to R_\infty, \\ (\dots, x_{-1}, x_0 \centerdot x_1, x_2, \dots) \mapsto (\dots, x_{-1} \centerdot x_0, x_1, x_2, \dots), \end{gathered}$$

shifts all the elements of the sequence one coordinate the right (i.e. it shifts the distinguished mark to the left).

Note that a general BSS machine without shift nodes is essentially a finite-dimensional BSS machine. It can store sequences of arbitrary length; however, without shift nodes it cannot access them. Thus, the strength of general machines comes from the combination of infinite machine spaces and the fifth type of node.

These two changes also enable general BSS machines to simulate classical Turing machines. The distinguished mark (the symbol " **.** ") in elements of R_∞ of a general BSS machine corresponds to the position of the read-write head of a Turing machine and the shift nodes simulate the shift operations of the read-write head, i.e. the instructions "move left" and "more right".

All notions defined for finite dimensional machines (for example computation path, halting set, input-output map) can be extended to general machines. In addition, a variant of Theorem 2.14 (the Path Decomposition Theorem) holds for general machines. We do not introduce this theory here since we do not need it in the rest of our work. However, it can be found in the book [1, Chapter 3].

2.5 Complexity theory in the BSS model

As in the classical (Turing) model of computation, we can talk about the complexity of a general BSS machine and consequently define complexity classes.

By complexity of a function or a map we intuitively mean the cost of computing it by a machine. The term "cost" typically refers to the optimal number of elementary operations of a machine or the optimal amount of space which is needed during the computation.

In the classical model of computation, complexity is expressed as a function of size of the input. Thus, before we formalize the notions "cost of computation" and "complexity of a map" in the BSS model of computation, we need to specify what we mean by "size of an input".

We introduce the theory of complexity in the BSS model only briefly since the direction of our work is different. In this section we follow the approach of authors of the book [1] in which more details can be found.

Let R be an arbitrary commutative ring with unit. Suppose that we have a *hight function* ht_R defined on R, i.e. a function that maps elements of R to non-negative integers. For a vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ we define the hight function as

$$\mathrm{ht}_R(x) = \max_{i \in \{1,\ldots,n\}} \mathrm{ht}_R(x_i).$$

In this work we consider the following hight functions

$$\begin{aligned}
\forall x \in \mathbb{R} \colon \mathrm{ht}_{\mathbb{R}}(x) &= 1,\\
\forall x \in \mathbb{Z}_2 \colon \mathrm{ht}_{\mathbb{Z}_2}(x) &= 1,\\
\forall x \in \mathbb{Z} \colon \mathrm{ht}_{\mathbb{Z}}(x) &= \lceil \log(|x| + 1) \rceil.
\end{aligned}$$

Recall from Section 2.4 that for $x \in R^n \subset R^\infty$ we define

$$\mathrm{length}(x) = n.$$

We define the *size of* $x \in R^n \subset R^\infty$ as

$$\mathrm{size}(x) = \mathrm{length}(x) \cdot \mathrm{ht}_R(x).$$

Over $\mathbb{R}$ an $\mathbb{Z}_2$ we consider the unit hight; thus, size of an input x is equal to its dimension. Over $\mathbb{Z}$ we consider the logarithmic hight; hence, size of an input x is approximately the bit length of the sequence x.

Now we are prepared to define what we mean by cost of computation. For $x \in R^n \subset R^\infty$ we define

$$\text{cost}_M(x) = T_M(x) \cdot \text{ht}_{max}(x),$$

where $T_M(x)$ is the halting time of the machine M on an input x and $\text{ht}_{max}(x)$ is the maximal hight of any element occurring in the computation of M on the input x. For machines over $\mathbb{R}$ and $\mathbb{Z}_2$, the cost of computation and the halting time of a machine M on an input x are the same.

A machine M over R is a *polynomial time machine* on $\mathbf{X} \subset R^\infty$ if

$$\exists c, q \in \mathbb{N}\ \forall x \in \mathbf{X}\colon \text{cost}_M(x) \leq c(\text{size}(x))^q.$$

The authors of the book [1] pay a lot of attention to complexity theory in the BSS model. As already mentioned in the beginning of this section, we can define complexity classes over an arbitrary ring with unit in the BSS model of computation. As one would expect, many interesting questions regarding their relationships arise. Is $\text{P}_\mathbb{R} = \text{NP}_\mathbb{R}$ true? Does $\text{P}_\mathbb{C} = \text{NP}_\mathbb{C}$ hold? Are these two questions related to each other? Would an answer to one of these questions tell us anything about the famous $\text{P} = \text{NP}$ problem in the classical model of computation?

These and many other questions were intensively studied especially in the early 1990s. And some of them were answered. For example, in 1991 Felipe Cucker proved that $\text{P}_\mathbb{R} \neq \text{NC}_\mathbb{R}$, see [5] for more details. However, it is a widely recognized conjecture that results from the complexity theory over real numbers do not bring an insight into the famous problems of the classical complexity theory.

At the end of this section, let us discuss the time complexity of finite dimensional BSS machines, which will be of our main interest in the rest of this work. Finite dimensional machines have a fixed dimension of the input space so the size of an input is fixed (bounded). Therefore, it makes little sense to talk about complexity classes for finite dimensional machines. However, it is still interesting to study the time complexity of these machines. Recall that the BSS machine defined in Example 2.3 is a finite dimensional machine whose halting time is not bounded. Thus, it make sense to ask how the halting time of a machine depends on the input of the machine.

Let us summarize. For finite dimensional machines we distinguish the following three situations:

1. The map is not computable by a finite dimensional machine.
2. The map is computable by a finite dimensional machine but not in constant time.
3. The map is computable by a finite dimensional machine in constant time.

We will see examples of all three kinds of maps later in the work.

2.6 Machines with built-in constants

In this work we will often study machines that, given an input, have to output an ϵ-approximation of some value that was derived from the input. More precisely. Let $\epsilon > 0$ be a fixed constant and $f\colon \mathbf{A} \to \mathbf{B}$, where $\mathbf{A}, \mathbf{B} \subseteq \mathbb{R}$, a real function. A BSS machine that approximates the function f with the precision ϵ is given an input $x \in \mathbf{A}$ and it has to output a value $y \in \mathbf{B}$ such that

$$|y - f(x)| < \epsilon.$$

Example 2.27. Let $\epsilon = 1$ and $f\colon \mathbb{R}^+ \to \mathbb{R}^+$ defined as $f(x) = \sqrt{x}$. Then a BSS machine computing the square root function with the precision $\epsilon = 1$ is a machine that, given $x \in \mathbb{R}^+$, outputs $y \in \mathbb{R}^+$ such that

$$|y - \sqrt{x}| < 1.$$

Note that the precision $\epsilon > 0$ is fixed; hence, it does not depend on the input y. Therefore, it is natural to understand the precision ϵ as a machine constant. Recall that every BSS machine has a set of machine constants. These constants appear as coefficients of the maps corresponding to the nodes of the machine.

Another way how to understand the precision ϵ is to consider it as "a part of hardware". Let us discuss this alternative point of view in more detail and explain what we mean by the expression " a part of hardware".

Let us first recall that for every real number $\epsilon \in \mathbb{R}^+$ there exists a rational number $\epsilon' \in \mathbb{Q}^+$ such that the inequalities

$$0 < \epsilon' \leq \epsilon$$

hold. It follows that we can restrict ourselves to rational $\epsilon > 0$. The next lemma explains why we are interested in this simplification.

Lemma 2.28. *Let $\epsilon \in \mathbb{Q}^+$ be fixed. Then for every elementary BSS machine M such that $\epsilon \in \mathbf{C}_M$ there exists an elementary BSS machine M' such that*

1. *the machines M and M' has the same machine constants except for ϵ which is not a machine constant of the machine M', i.e.*

$$\mathbf{C}_{M'} = \mathbf{C}_M \setminus \{\epsilon\};$$

2. *the machines M and M' have the same input-output map and the same halting set; in particular,*

$$\Phi_M = \Phi_{M'} \text{ and } \Omega_M = \Omega_{M'};$$

3. *the time complexities of both machines are proportionally the same, i.e. for every $x \in \Omega_M = \Omega_{M'}$ we have*

$$T_M(x) \leq T_{M'}(x) \leq T_M(x) + \mathcal{O}(1).$$

Proof. Let $p, q \in \mathbb{N}$ be smallest possible natural numbers such that $\epsilon = \frac{p}{q}$. We know that such natural numbers exist since $\epsilon \in \mathbb{Q}^+$. The natural numbers p, q can be expressed as

$$p = \underbrace{1 + \cdots + 1}_{p}, \quad q = \underbrace{1 + \cdots + 1}_{q}. \tag{2.10}$$

Recall that we assume that every BSS machine has 0 and 1 as machine constants. Thus, from (2.10) we see that the value ϵ can be computed in $p + q + 1$ elementary instructions from the constant 1. Note that this is an upper bound on the number of elementary instructions since there might be a more efficient way how to compute the values p, q from the constant 1.

We are given a BSS machine M and we construct a BSS machine M' in the following way. All appearances of the constant ϵ in the input node of the machine M are replaced by the constant 0. Then the machine M' performs the following sequence of $p + q + 4$ elementary instructions

$$\begin{array}{ll}
(s_1, s_2, s_3) \leftarrow (1, 1, 1); & (s_1, s_2, s_3) \leftarrow (s_1, s_2 + s_3, s_3); \\
(s_1, s_2, s_3) \leftarrow (s_1 + s_3, s_2, s_3); & \qquad \vdots\, q \\
\qquad \vdots\, p & (s_1, s_2, s_3) \leftarrow (s_1, s_2 + s_3, s_3); \\
(s_1, s_2, s_3) \leftarrow (s_1 + s_3, s_2, s_3); & (s_1, s_2, s_3) \leftarrow ({}^{s_1}/{}_{s_2}, s_2, s_3).
\end{array}$$

Note that the first instruction represents in fact three elementary instructions. The machine M' stores the computed value ϵ in one component of the machine state, let us denote this component s'. Thereafter, the machine M' performs the elementary instruction $s \leftarrow s'$ for every component s of the machine state which was assigned by the constant 0 instead of the constant ϵ in the input node. Observe that there are at most m instructions of this kind needed, where m is the dimension of the state space of the machine M.

Now the machine M' copies all nodes of the machine M. When a computation node performing the instruction $s \leftarrow \epsilon$ occurs, the machine M' replaces it with the instruction $s \leftarrow s'$.

It follows from our construction that $\mathbf{C}_{M'} = \mathbf{C}_M \setminus \{\epsilon\}$. The second part of the lemma follows from the fact that the only elementary instructions that were changed do not influence the input-output map or the halting set. The machine M' has at most $(p + q + 4) + m$ additional elementary instructions, i.e. for every $x \in \Omega_M = \Omega_{M'}$ we have

$$T_M(x) + (p + q + 4) + m = T_{M'}(x).$$

Since the value ϵ is fixed, the values p, q are fixed as well. Since M is a finite dimensional machine, the dimension m is also fixed. The third part of the lemma follows. $\square$

We say the BSS machine M' from the proof above has ϵ as a *built-in constant.*

Based on the previous discussions, let us establish the following convention. In the rest of this work, when we talk about a BSS machine which has a fixed precision $\epsilon > 0$ then

1. we assume that $\epsilon \in \mathbb{Q}^+$;

2. the BSS machine has ϵ as a built-in constant. In particular, ϵ is not in the set of the machine constants but it is computed from the constant 1 at the beginning of the computation.

At the end of this section, let us discuss what we understand by the time complexity of an approximation problem (such as for example the approximation of a zero of a real polynomial).

Intuitively, when we study the time complexity of an approximation problem, we expect it to depend on the input value and the required precision of our approximation. However, this does not match with the theory which we introduced in this section. Observe that we were taking about BSS machines that have a fixed precision $\epsilon > 0$. Time bounds may depend on ϵ but we shall treat ϵ as a fixed parameter, not as a variable.

Therefore, if we want to study the time complexity of an approximation problem then we implicitly consider a family of BSS machines $\mathcal{M} = \{M_\epsilon\}_{\epsilon \in \mathbb{Q}^+}$. All machines in the family $\mathcal{M}$ solve the same approximation problem but each of them with a different precision. Hence, it make sense to talk about the general time complexity of the family, which is now a function of the input and the precision. Note that the precision determines which BSS machine in the family we refer to.

2.7 Informal algorithms

Let us close this chapter about BSS machines with a technical convention. We introduced the theory of BSS machines to formally capture algorithms which operates with real numbers. However, it is impractical to describe every algorithm formally as a BSS machine. Therefore, in the rest of the work we write down algorithms informally in "pseudo-code". It should be always clear that the informal algorithm can be formalized as a BSS machine.

So for example the BSS machine defined in Example 2.3 would be informally describe by Algorithm 2.1.

We will also frequently use the informal notion "while-loop". We believe that the meaning of this expression is intuitively clear. In this concrete example, by one "while-loop" we understand the sequence of nodes $(2, 3)$, i.e. the sequence (branch node, computation node).

```
Input: $x \in \mathbb{R}$
$k \leftarrow 0$;
while $x \geq 1$ do
    $x \leftarrow x - 1$;
    $k \leftarrow k + 1$;
end
return $k$
```

Algorithm 2.1: Pseudo-code of the BSS machine defined in Figure 2.1, which computes the floor function for non-negative real numbers

3. The square root function

Our main aim is to analyse the functions from $\mathbb{R}$ to $\mathbb{R}$ which are easy to compute with a BSS machine but hard to invert. In the previous chapter we showed that BSS machines compute maps that are piecewise rational. Therefore, polynomials and rational functions over $\mathbb{R}$ are of our interest.

Let us first assume that we have a non-zero rational function ψ which can be represented by two polynomials $p, q \in \mathbb{R}[x]$ which are both of degree less or equal to one. The rational function ψ is certainly easy to compute with a BSS machine. Since ψ is a non-zero function, the polynomial p must be non-zero as well. Let φ be a rational function such that $\varphi = \frac{q}{p}$. Then clearly $\psi \cdot \varphi = 1$. Moreover, the rational function φ is as easy to compute in the BSS model as the rational function ψ.

This discussion explains why it makes no sense to consider linear polynomials or rational functions which can be represented by a fraction of two linear polynomials. We are naturally led to study polynomials of higher degree.

In this chapter, we focus on BSS machines which try to invert the most simple non-linear polynomial, i.e. the polynomial $x^2 \in \mathbb{R}[x]$. In other words, we are interested in BSS machines which compute the square root of a non-negative real number. In Section 3.2 we study if there exists a BSS machine which computes the square root function with infinite precision. In Section 3.3 we focus on BSS machines which compute an ϵ-approximation of the square root function, where $\epsilon > 0$ is an arbitrary but fixed constant.

Before we start, let us introduce one important convention about notation.

3.1 Notation

In the rest of this work M_0 denotes the BSS machine which we are trying to invert and M denotes a candidate for an inverting machine. The input of the machine M_0 will be denoted by x and the output will be denoted by y. Therefore, we understand the input-output map of the machine M_0 as a real function whose variable is x and the input-output map of the machine M as a real function whose variable is y. This convention is illustrated in Figure 3.1.

Hence, in this chapter M_0 is a BSS machine that computes the square function, i.e. $\Phi_{M_0}(x) = x^2$ and $\Omega_{M_0} = \mathbb{R}$. In Section 3.2 we study whether there exists a BSS machine M such that $\Omega_M = \mathbb{R}^{\geq 0}$ and $\Phi_M(y) = \sqrt{y}$. In this case we have the following equivalence

$$\Phi_M(y) = \sqrt{y} \Leftrightarrow (\Phi_M(y))^2 = y.$$

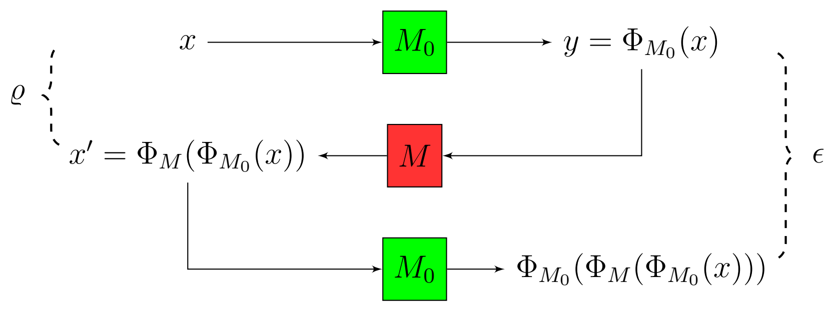
Consequently, it does not matter if we study the problem of *inverting the square function* or the problem of *computing the square root function*.

We have to be more careful in Section 3.3. Note that if there exists a BSS machine M such that for every $y \in \mathbb{R}^{\geq 0}$

$$|\Phi_M(y) - \sqrt{y}| < \epsilon,$$

where $\epsilon > 0$ is the fixed precision, then there also exists a BSS machine M' such that for every $y \in \mathbb{R}^{\geq 0}$

$$|(\Phi_{M'}(y))^2 - y| < \epsilon.$$



x
M_0
$y = \Phi_{M_0}(x)$
ϱ
$x' = \Phi_M(\Phi_{M_0}(x))$
M
ϵ
M_0
$\Phi_{M_0}(\Phi_M(\Phi_{M_0}(x)))$

Figure 3.1: Illustration of the used notation

Since the square function is injective on $\mathbb{R}^{\geq 0}$, the opposite holds as well. So if we are interested only in the existence of a computing (respectively, an inverting) machine then it does not matter which of these two points of view we consider. However, for a fixed precision $\epsilon > 0$, the time complexity of computing the square root function with the precision ϵ is not the same as the time complexity of inverting the square function with the precision ϵ. To distinguish these two complexities and avoid confusion, we will use the letter ϱ in the first case and the letter ϵ in the second case. It is not difficult to prove that for $y \in \mathbb{R}^{\geq 0}$ and $\epsilon > 0$ the implication

$$|\Phi_M(y) - \sqrt{y}| < \varrho \Rightarrow |(\Phi_M(y))^2 - y| < \epsilon$$

holds if $0 < \varrho \leq \sqrt{y + \epsilon} - \sqrt{\epsilon}$.

Figure 3.1 summarizes the established convention.

3.2 A square root with infinite precision

In this section we study the following problem. Is there a BSS machine which, given an input $y \in \mathbb{R}^{\geq 0}$, outputs a value $x' \in \mathbb{R}$ such that $(x')^2 = y$?

Let us first focus on constant time BSS machines.

Theorem 3.1. *The set*

$$\{y \in \Omega_M \colon (\Phi_M(y))^2 = y\}$$

is finite for every constant time BSS machine M.

Proof. Without lost of generality we can assume that M is an elementary BSS machine (see Observation 2.20 for more details).

Let $\gamma \in \Gamma_M$ be an arbitrary halting path and consider the set

$$\{y \in \nu_\gamma \colon (\varphi_\gamma(y))^2 = y\}.$$

We define a new polynomial $\psi_\gamma \in \mathbb{R}[y]$ as

$$\psi_\gamma(y) = p_\gamma^2(y) - q_\gamma^2(y) \cdot y,$$

where $p_\gamma, q_\gamma \in \mathbb{R}[y]$ are such that $\varphi_\gamma = \frac{p_\gamma}{q_\gamma}$. Observe that

$$\deg(p_\gamma^2(y)) \neq \deg(q_\gamma^2(y) \cdot y)$$

since the value $\deg(p_\gamma^2(y))$ is even whereas the value $\deg(q_\gamma^2(y)\cdot y)$ is odd.

Recall that M is a constant time machine. Thus, there exists a natural number $T\in\mathbb{N}$ such that $T_M(y)\le T$ for every $y\in\Omega_M$. Consequently, the number of elementary instructions performed by the machine M on an input $y\in\Omega_M$ is also bounded by the value T. Then we know by Theorem 2.23 that

$$\deg(p_\gamma)\le 2^T \quad\text{and}\quad \deg(q_\gamma)\le 2^T.$$

Thus, we derived the following bound

$$\deg\psi_\gamma=\max\{2\deg(p_\gamma),2\deg(q_\gamma)+1\}\le 2^{T+1}+1.$$

Since the rational function φ_γ is defined on ν_γ, we have the following equality of sets

$$\{y\in\nu_\gamma\colon(\varphi_\gamma(y))^2=y\}=\{y\in\nu_\gamma\colon\psi_\gamma(y)=0\}.$$

We can bound the cardinality of these sets using Lemma 1.15 as

$$|\{y\in\nu_\gamma\colon(\varphi_\gamma(y))^2=y\}|=|\{y\in\nu_\gamma\colon\psi_\gamma(y)=0\}|\le 2^{T+1}+1.$$

The Path Decomposition Theorem implies the equality

$$|\{y\in\Omega_M\colon(\Phi_M(y))^2=y\}|=\sum_{\gamma\in\Gamma_M}|\{y\in\nu_\gamma\colon(\varphi_\gamma(y))^2=y\}|.$$

We know from Lemma 2.5 that $|\Gamma_M|\le 2^T$. Consequently, we have

$$|\{y\in\Omega_M\colon(\Phi_M(y))^2=y\}|\le 2^T\cdot(2^{T+1}+1).$$

Hence, we proved that the set $\{y\in\Omega_M\colon(\Phi_M(y))^2=y\}$ is finite. □

Let us now state one important consequence of the theorem.

Corollary 3.2. *No BSS machine computes the real function $\sqrt{y}$ exactly on any non-trivial subinterval of $\mathbb{R}^{\ge 0}$.*

Proof. Let M be an arbitrary BSS machine and let $T\in\mathbb{N}$. From Theorem 3.1 we know that the set

$$\{y\in\Omega_{M,T}\colon(\Phi_{M,T}(y))^2=y\}$$

is finite. Hence,

$$\{y\in\Omega_M\colon(\Phi_M(y))^2=y\}=\bigcup_{T\in\mathbb{N}}\{y\in\Omega_{M,T}\colon(\Phi_{M,T}(y))^2=y\}$$

is a countable union of finite set which means that it is a countable set. However, every non-trivial subinterval of $\mathbb{R}$ is uncountable. □

Does it mean that the square function is an “easy to compute and hard to invert function”? There is only one elementary operation needed in order to compute x^2 but, as we just proved, there is no BSS machine that can compute the inverse. So the answer to the question can be “Yes”.

Nevertheless, the situation in which we allow the inverting machine to make a small mistake seems to be more natural when working with real numbers. Recall that one of our motivations is biometric authentication. We have a biological trait which changes a little over time. Then the requirement of equality with infinite precision does not make a lot of sense. Mainly, because we would not be able to authenticate ourselves.

Let us therefore study the case in which a small error is allowed.

3.3 A square root with an arbitrary but fixed precision

We are now in the situation in which $\varrho > 0$ is a fixed constant and we study if there is a BSS machine M such that for every $y \in \mathbb{R}^{\geq 0}$ it holds

$$|\Phi_M(y) - \sqrt{y}| < \varrho.$$

The reason why we now use ϱ instead of ϵ is explained in Section 3.1.

We prove in Subsection 3.3.1 that no constant time machine can approximate $\sqrt{y}$ with the precision ϱ for all $y \in \mathbb{R}^{\geq 0}$. Then a natural question arises; is there any BSS machine that computes the square root function with the precision ϱ? And if so, what is its time complexity?

We show that there is a positive answer to the first question. To answer the latter, we present both an upper bound (in Subsection 3.3.2) and a lower bound (in Subsection 3.3.3) on the time complexity of this problem.

3.3.1 Constant time machines

Let $\varrho > 0$ be an arbitrary fixed constant. Firstly, we prove that for every constant time BSS machine M the set

$$\{y \in \mathbb{R}^{\geq 0} \colon |\Phi_M(y) - \sqrt{y}| < \varrho\}$$

is bounded. As a consequence, no constant time machine can compute the square root function for all non-negative real numbers.

Theorem 3.3. *Let $\varrho > 0$. For every constant time BSS machine M there exists $\delta > 0$ such that for every $y > \delta$ it holds that $|\Phi_M(y) - \sqrt{y}| > \varrho$.*

Let us first observe that by the definition of limit it is sufficient to show that

$$\lim_{y \to \infty} |\Phi_M(y) - \sqrt{y}| = \infty.$$

Before we prove the theorem, let us state an auxiliary lemma about limits.

Lemma 3.4. *Let $a, b \in \mathbb{R}[y]$ be two polynomials such that*

$$a(y) = \sum_{i=0}^{n} a_i y^i, \quad b(y) = \sum_{i=0}^{m} b_i y^i,$$

where $a_i, b_i \in \mathbb{R}, n, m \geq 0$ and $a_n, b_m \neq 0$. Then

$$\lim_{y \to \infty} \left| \frac{a(y)}{b(y)} - \sqrt{y} \right| = \infty.$$

Proof. a) If $n > m$ then by the Algebraic Limit Theorem we have

$$\begin{aligned} \lim_{y \to \infty} \frac{a(y) - b(y) \cdot \sqrt{y}}{b(y)} &= \lim_{y \to \infty} \left(\frac{y^n}{y^m} \cdot \frac{\frac{a(y)}{y^n} - \frac{b(y) \cdot \sqrt{y}}{y^n}}{\frac{b(y)}{y^m}} \right) \\ &= \frac{a_n}{b_m} \cdot \lim_{y \to \infty} y^{n-m}, \end{aligned}$$

which equals $\pm\infty$ depending on the sign of $\frac{a_n}{b_m}$. Recall that $a_n, b_m \neq 0$.

b) If $n \leq m$ then again by the Algebraic Limit Theorem we have

$$\lim_{y\to\infty} \frac{a(y) - b(y)\cdot\sqrt{y}}{b(y)} = \lim_{y\to\infty} \left(\frac{y^m \cdot \sqrt{y}}{y^m} \cdot \frac{\frac{a(y)}{y^m\cdot\sqrt{y}} - \frac{b(y)\cdot\sqrt{y}}{y^m\cdot\sqrt{y}}}{\frac{b(y)}{y^m}} \right)$$
$$= \frac{-b_m}{b_m} \cdot \lim_{y\to\infty} \sqrt{y}$$

which equals $-\infty$.

We proved that $\lim_{y\to\infty} \left(\frac{a(y)}{b(y)} - \sqrt{y} \right) = \pm\infty$ and the statement follows. $\square$

Now we are prepared to prove Theorem 3.3.

Proof (Theorem 3.3). Let M be a constant time BSS machine. By definition there is a natural number $T \in \mathbb{N}$ such that $T_M(y) \leq T$ for all $y \in \Omega_M$. We know from Theorem 2.14 that φ_γ is a rational function for every $\gamma \in \Gamma_M$. So according to Lemma 3.4 there exists $\delta_\gamma > 0$ such that for all $y \in \nu_\gamma$ the following implication holds

$$y > \delta_\gamma \Rightarrow |\varphi_\gamma(y) - \sqrt{y}| > \varrho.$$

Let us define the value δ as

$$\delta := \max_{\gamma\in\Gamma_M} \delta_\gamma.$$

By Lemma 2.5 we have the bound $|\Gamma_M| \leq 2^T$. Consequently, the set $\{\delta_\gamma \colon \gamma \in \Gamma_M\}$ is finite and thus the maximum exists.

Next, fix an arbitrary real number $y > \delta$. Then by Theorem 2.14 the equality

$$|\Phi_M(y) - \sqrt{y}| = |\varphi_\gamma(y) - \sqrt{y}|$$

holds for exactly one $\gamma \in \Gamma_M$. Because $y > \delta \geq \delta_\gamma$, we derived that

$$|\Phi_M(y) - \sqrt{y}| > \varrho.$$

$\square$

3.3.2 Upper bound on the time complexity

The aim of this section is to construct a concrete BSS machine which approximates $\sqrt{y}$ with the precision ϱ, where $\varrho > 0$ is fixed. Secondly, we bound the time complexity of the constructed machine. And finally, we analyse the set of machine constants.

In fact, we present two algorithms (BSS machines) in this section. The first one is essentially the half-interval search algorithm which has the time complexity $\mathcal{O}(\log_2(y) - \log_2(\varrho))$. The second algorithm is based on Newton's method which has a better time complexity $\mathcal{O}(\log_2(\log_2(y) - \log_2(\varrho)))$. Nevertheless, Newton's method does not always converge. In order to solve this problem, we first have to make some non-trivial precomputations. In total, we achieve the time complexity

$$\mathcal{O}\left((\log_2(\log_a(y) - \log_a(\varrho)))^2\right),$$

where $a \approx 1.6$.

In sake of proving the claimed time complexities, we need to include many technical and rather lengthy computations in this subsection.

Before we start with the first method, let us recall some technical conventions from the previous chapter. An algorithm (written down in "pseudo code") will informally describe a family of BSS machines $\mathcal{M} = \{M_\varrho\}_\varrho$. A BSS machine $M_\varrho \in \mathcal{M}$ approximates the square root function with the precision ϱ. Moreover, a machine M_ϱ has ϱ as a built-in constant. And finally, when we talk about the time complexity of the algorithm then we mean the time complexity of the corresponding family of machines (i.e. it depends both on the input y and the precision ϱ). See Section 2.6 and Section 2.7 for more details.

The half-interval search method

Let $\varrho > 0$ be fixed. The idea of this algorithm is very simple. We start with a closed interval which contains $\sqrt{y}$. In every step we take the middle of the interval and decide which subinterval contains $\sqrt{y}$. Then we apply the same procedure on this subinterval. We continue until the length of the interval is smaller than ϱ. We end up with an interval that contains $\sqrt{y}$ and every point of the interval approximates $\sqrt{y}$ with the precision ϱ.

```
Input: $y \in \mathbb{R}^{\geq 0}$
Output: $z \in \mathbb{R}$ such that $|z - \sqrt{y}| < \varrho$
$z_L \leftarrow 0$;
$z_R \leftarrow \max\{1, y\}$ ;
while $z_R - z_L \geq \varrho$ do
    $z_M \leftarrow z_L + \frac{z_R - z_L}{2}$;
    if $z_M^2 < y$ then
        $z_L \leftarrow z_M$;
    end
    else
        $z_R \leftarrow z_M$;
    end
end
return $z_L$
```

Algorithm 3.1: Approximation of $\sqrt{y}$ based on the half-interval search method

Theorem 3.5. *Algorithm 3.1 halts on every valid input $y \in \mathbb{R}^{\geq 0}$ and it outputs a value $z \in \mathbb{R}$ such that*

$$|z - \sqrt{y}| < \varrho.$$

Its time complexity for $y \geq 1$ is $\mathcal{O}\left(\log_2\left(\frac{y}{\varrho}\right)\right)$ while the set of the constants is $\{0, 1\}$.

Proof. Let us denote $z_L^{(m)}, z_R^{(m)}, z_M^{(m)}$ the values z_L, z_R, z_M after the m-th while-loop. By induction on m we first prove that

$$z_L^{(m)} \leq \sqrt{y} \leq z_R^{(m)},$$

for every $m \geq 0$.

For $m = 0$ we have to distinguish two cases. If $y \geq 1$ then $0 \leq \sqrt{y} \leq y$. This is what we want since $z_R^{(0)} = y$. The other case follows from the fact that if $0 \leq y < 1$ then also $0 \leq \sqrt{y} < 1$.

Let us now assume that the inequalities $z_L^{(m)} \leq \sqrt{y} \leq z_R^{(m)}$ hold. According to the algorithm, the value $z_M^{(m+1)}$ is computed as

$$z_M^{(m+1)} \leftarrow z_L^{(m)} + \frac{z_R^{(m)} - z_L^{(m)}}{2}.$$

Since the square root function is strictly increasing, the following implication holds

$$\left(z_M^{(m+1)}\right)^2 < y \Rightarrow z_M^{(m+1)} < \sqrt{y}.$$

By setting $z_L^{(m+1)} \leftarrow z_M^{(m+1)}$ and $z_R^{(m+1)} \leftarrow z_R^{(m)}$ we achieve

$$z_L^{(m+1)} = z_M^{(m+1)} < \sqrt{y} \overset{IH}{\leq} z_R^{(m)} = z_R^{(m+1)}.$$

Analogously, the implication

$$\left(z_M^{(m+1)}\right)^2 \geq y \Rightarrow z_M^{(m+1)} \geq \sqrt{y}$$

holds. By assigning $z_L^{(m+1)} \leftarrow z_L^{(m)}$ and $z_R^{(m+1)} \leftarrow z_M^{(m+1)}$ we have

$$z_L^{(m+1)} = z_L^{(m)} \overset{IH}{\leq} \sqrt{y} \leq z_M^{(m+1)} = z_R^{(m+1)}.$$

We proved that $\sqrt{y} \in \left[z_L^{(m)}, z_R^{(m)}\right]$ for every $m \geq 0$.

According to the algorithm, we proceed until $z_R^{(m)} - z_L^{(m)} < \varrho$. Then for any $z \in \left[z_L^{(m)}, z_R^{(m)}\right]$ it holds that $|z - \sqrt{y}| < \varrho$ which implies the correctness of the algorithm.

Note that for every $m \geq 0$ we have

$$z_R^{(m+1)} - z_L^{(m+1)} = \frac{z_R^{(m)} - z_L^{(m)}}{2}.$$

Consequently, the following equality holds

$$z_R^{(m)} - z_L^{(m)} = \frac{z_R^{(0)} - z_L^{(0)}}{2^m}.$$

For $y < 1$, we can assume that $\varrho < 1$ (otherwise we can just output y). In this case the algorithm stops when

$$\frac{1}{2^m} < \varrho.$$

This means that the algorithm stops after $m = \lceil -\log_2(\varrho) \rceil$ while-loops.

When $y \geq 1$, the initial difference $z_R^{(0)} - z_L^{(0)}$ equals y. So the algorithm stops in this case when

$$\frac{y}{2^m} < \varrho \;\Leftrightarrow\; \log_2(y) - \log_2(\varrho) < m.$$

Thus the number of while-loops equals $\lceil \log_2(y) - \log_2(\varrho) \rceil$.

For studying the time complexity, we are interested only in the case in which $y \geq 1$. From the discussion above we know the number of performed while-loops. There are only constantly many elementary instructions (computation nodes) and one branch node needed in every while-loop. Thus, we can conclude that the time complexity of the algorithm is $\mathcal{O}\left(\log_2\left(\frac{y}{\varrho}\right)\right)$ as stated in the theorem.

Finally, note that the constant 2 can be computed by one elementary instruction from the constant 1 and recall that we assume ϱ to be a built-in constant. As a result, the set of machine constants is the set $\{0, 1\}$. $\square$

We proved that there exists a very simple BSS machine that approximates the square root for any $y \in \mathbb{R}^{\geq 0}$ with some fixed accuracy $\varrho > 0$. Next, we try to find a BSS machine which does the same but more efficiently.

Newton's method

It will be useful to rephrase our problem in the following way. Approximate the positive root of the polynomial $f(z) = z^2 - y \in \mathbb{R}[z]$ with precision $\varrho > 0$. This equivalent task is, indeed, very well know and studied. There are many methods in numerical analysis for root approximation of a real polynomial.

One of these methods is Newton's method (know also as the Newton-Raphson method). Authors of the book [1] pay a lot of attention to Newton's algorithm; mainly, in Chapters 8 and 9. Because we base our second algorithm on this method, we introduce its main ideas and state important theorems. Nevertheless, technical proofs are omitted and can be found in the book mentioned above.

Definition 3.6. Given a one-variable polynomial f over $\mathbb{R}$ we define the *Newton endomorphism* $N_f\colon \mathbb{R} \to \mathbb{R}$ as

$$N_f(z) = z - \frac{f(z)}{f'(z)}.$$

As usual, the f' denotes the first derivative of the function f. The Newton endomorphism is defined if $f'(z) \neq 0$.

Let us establish the following notation. For $k \in \mathbb{N}$ we denote

$$z_k = N_f^{(k)}(z_0) = (\underbrace{N_f \circ \cdots \circ N_f}_{k})(z_0).$$

The idea is to choose a good starting point $z_0 \in \mathbb{R}$ and apply the Newton endomorphism iteratively until we find a point that is ϱ close to a zero of the polynomial f. Algorithm 3.2 makes this idea more formal.

The problem is that Algorithm 3.2 is generally not convergent. Let us demonstrate it on an example.

```
Input: $z_0 \in \mathbb{R}$
Output: $z \in \mathbb{R}$ such that $|f(z)| < \varrho$
$z \leftarrow N_f(z_0)$;
while $|f(z)| > \varrho$ do
    $z \leftarrow N_f(z)$;
end
return $z$
```

Algorithm 3.2: Newton's algorithm

Example 3.7 ([1, Section 1.2.3]). For the polynomial $f(z) = z^3 - 2z + 2 \in \mathbb{R}[z]$ we have

$$N_f(z) = z - \frac{z^3 - 2z + 2}{3z^2 - 2}.$$

If we choose $z_0 = 0$ then $z_1 = N_f(z_0) = 1$ and $z_2 = N_f(z_1) = 0$. We see that $z_i \in \{0, 1\}$ for every $i \in \mathbb{N}$. Thus, the sequence $(z_i)_{i=0}^{\infty}$ does not converge to any root of the polynomial f.

This shows that there are bad starting points for which Newton's method does not work. It turns out that if we start sufficiently close to a zero of the polynomial then the sequence $(z_i)_{i=0}^{\infty}$ converges. We formalize what we mean by "sufficiently close" in the next definition.

Definition 3.8 ([1, Definition 1 in Chapter 8]). We call $z \in \mathbb{R}$ an *approximate zero* of a polynomial f if the sequence $z_0 = z, z_i = N_f(z_{i-1})$ is defined for all $i \in \mathbb{N}$ and there is a real number ξ such that $f(\xi) = 0$ and

$$|z_i - \xi| \leq \left(\frac{1}{2}\right)^{2^i - 1} |z - \xi|.$$

We call ξ the *associate zero.*

The following lemma studies the time complexity of Newton's algorithm assuming that the input is an approximate zero.

Lemma 3.9. *Let $z \in \mathbb{R}$ be an approximate zero of a polynomial f with associated zero ξ. Let $L \in \mathbb{R}$ be such that $|z - \xi| \leq L$. Then we can approximate the zero ξ to any accuracy $\varrho > 0$ in*

$$\lceil \log(|\log(\varrho)| + \log(L) + 1) \rceil$$

iterations of the Newton endomorphism.

Proof. We want to find $i \in \mathbb{N}$ such that $|z_i - \xi| < \varrho$. Since $z_0 := z$ is an approximate zero and $|z - \xi| \leq L$, we know that

$$|z_i - \xi| \leq \left(\frac{1}{2}\right)^{2^i - 1} \cdot L.$$

It is sufficient to find $i \in \mathbb{N}$ such that

$$2^{-(2^i - 1)} L < \varrho \;\Leftrightarrow\; 2^i > -\log(\varrho) + \log(L) + 1.$$

Note that $-\log(\varrho) \leq |\log(\varrho)|$. Thus, it suffices to find $i \in \mathbb{N}$ such that

$$2^i > |\log(\varrho)| + \log(L) + 1 \quad \Leftrightarrow \quad i > \log\left(|\log(\varrho)| + \log(L) + 1\right).$$

□

A natural question arises. How to find an approximate zero of a polynomial? This is, indeed, a non-trivial question on which we concentrate in the rest of this subsection.

Following the approach of authors of the book [1], let us first define auxiliary quantities $\gamma(f,z), \beta(f,z)$ and $\alpha(f,z)$. For a polynomial f and a point $z \in \mathbb{R}$ we define

$$\gamma(f,z) = \sup_{k \geq 2} \left| \frac{f^{(k)}(z)}{f'(z) \cdot k!} \right|^{\frac{1}{k-1}}, \tag{3.1}$$

$$\beta(f,z) = \left| \frac{f(z)}{f'(z)} \right|, \tag{3.2}$$

$$\alpha(f,z) = \beta(f,z) \cdot \gamma(f,z). \tag{3.3}$$

where $f^{(k)}$ denotes the k-th derivative of f as usual.

The following theorem is crucial for our algorithm since it gives a criterion for z to be an approximate zero.

Theorem 3.10 ([1, Theorem 2 in Chapter 8])**.** *There is a universal constant α_0 such that if $\alpha(f,z) < \alpha_0$ then z is an approximate zero of f. Moreover, the distance of z from the associated zero is at most $2 \cdot \beta(f,z)$.*

Remark 3.11 ([1, Remark 6 in Chapter 8]). A lot of research have been done in order to prove Theorem 3.10 with the constant α_0 as large as possible. The authors of the book [1] give a proof for $\alpha_0 = 0.03$. S. Smale proved in [11] that the theorem holds for a larger constant, i.e. $\alpha_0 \approx 0.130707$. Furthermore, the author of [14] improved the bound to $\alpha_0 \approx 0.157671$. For our purposes it is important that the constant exists, that it lies in the interval $\left(0, \frac{1}{4}\right)$ and that it is only constantly times smaller than 2.

We will now apply this general theory to our concrete problem. For a given real number $y \in \mathbb{R}^{\geq 0}$ and fixed precision $\varrho > 0$, we want to approximate the non-negative root of the polynomial $z^2 - y \in \mathbb{R}[z]$ with precision $\varrho > 0$. Let us therefore set

$$f(z) = z^2 - y.$$

At this point we should mention that this special case of Newton's method is also known as the Babylonian method. It is one of the oldest method of square root approximation. Interestingly, this method is still widely implemented and used nowadays.

Let us now apply the general theory from above on this special case. For $f(z) = z^2 - y$ we have

$$f^{(k)} = \begin{cases} 2z, & \text{for } k = 1; \\ 2, & \text{for } k = 2; \\ 0, & \text{for } k > 2. \end{cases}$$

The Newton endomorphism is in this case

$$N_f(z) = z - \frac{f(z)}{f'(z)} = z - \frac{z^2 - y}{2z} = \frac{z + \frac{y}{z}}{2}.$$

Note that it this special case, the Newton endomorphism is in fact the average of z and y/z.

We can also express the quantities γ, β, α. Using formulae (3.1), (3.2) and (3.3) on the polynomial $f(z) = z^2 - y$, we achieve

$$\gamma(f, z) = \left|\frac{f^{(2)}(z)}{f'(z) \cdot 2!}\right| = \frac{1}{2 \cdot |z|},$$
$$\beta(f, z) = \left|\frac{z^2 - y}{2z}\right|,$$
$$\alpha(f, z) = \frac{1}{2 \cdot |z|} \cdot \left|\frac{z^2 - y}{2z}\right| = \frac{|z^2 - y|}{4z^2}.$$

Finally, Theorem 3.10 gives us the condition

$$\frac{|z^2 - y|}{4z^2} < \alpha_0.$$

Thus, for $0 < \alpha_0 < \frac{1}{4}$ we have the following inequalities

$$\frac{y}{1 + 4\alpha_0} < z^2 < \frac{y}{1 - 4\alpha_0}.$$

We know from Remark 3.11 that the constant α_0 can be chosen from this interval. Note that for $y > 0$ both

$$\frac{y}{1 + 4\alpha_0} < y \quad \text{and} \quad y \cdot (1 + 4\alpha_0) < \frac{y}{1 - 4\alpha_0}$$

hold. It follows that it is sufficient to find z such that

$$y \le z^2 \le y \cdot (1 + 4\alpha_0). \tag{3.4}$$

Lemma 3.12. *Let $a \in \mathbb{R}$ be such that $1 \le a \le 1 + 4\alpha_0$. For $y \ge a$, let $b \in \mathbb{N}$ denote the maximal positive integer with the property $a^b \le y$. Then*

$$z = a^{\frac{b+1}{2}}$$

satisfies the inequalities (3.4).

Proof. The square of the z from the statement is $z^2 = a^{b+1}$. Thus, from the maximality of b we know that $y \le z^2$. To prove the other inequality, recall that $a \le 1 + 4\alpha_0$ which implies that

$$z^2 \le a^b \cdot (1 + 4\alpha_0).$$

Moreover, we know that $a^b \le y$. We derived that

$$z^2 \le y \cdot (1 + 4\alpha_0).$$

□

Lemma 3.12 tells us that, in order to find an approximate zero of the polynomial $f(z) = z^2 - y$, we need to

I.) find a constant a such that $1 \leq a \leq 1 + 4\alpha_0$ and then

II.) find the value $a^{\frac{b+1}{2}}$ where b is the maximal integer such that $a^b \leq y$.

We study each part separately. Let us first focus on the task I.). In fact we show how to construct a constant a using only constants from the set $\{0, 1\}$. It will be useful to have the value $\sqrt{a}$ as well (the reason becomes clear later). Therefore, our approach is to first construct a value c such that $1 \leq c^2 \leq 1 + 4\alpha_0$ and then set $a = c^2$.

The construction, indeed, depends on the constant α_0. According to Remark 3.11 the best known constant α_0 is approximately 0.157671. In that case $(1 + 4\alpha_0) \approx 1.630684$. Let us therefore introduce one concrete procedure that will output constants c and a such that $a \leq 1.630684$. The idea is to set $c = 1 + \frac{1}{2} \cdot \frac{1}{2} = 1.25$. Then $a = c^2 = 1.5625 < 1.630684$. Algorithm 3.3 describes the construction of c in detail as a sequence of elementary instructions.

```
$(s_1, s_2) \leftarrow (1, 0)$ ;
$(s_1, s_2) \leftarrow (s_1, s_1 + s_1)$ ;                    // $(1, 2)$
$(s_1, s_2) \leftarrow (s_1, 1/s_2)$ ;                        // $(1, 1/2)$
$(s_1, s_2) \leftarrow (s_1, s_2 \cdot s_2)$ ;                // $(1, 1/4)$
$(s_1, s_2) \leftarrow (s_1 + s_2, s_2)$ ;                    // $(5/4, 1/4)$
$(s_1, s_2) \leftarrow (s_1, s_1^2)$ ;                        // $(5/4, 25/16)$
**return** $s_1, s_2$
```

Algorithm 3.3: Construct the constants a and c

We see that the constants c and a can be constructed from constants $\{0, 1\}$ in seven elementary instructions (we need two instructions for the initialization $(s_1, s_2) \leftarrow (1, 0)$). It should be clear that a similar BSS machine can be built even if a different constant α_0 is considered.

Now we focus on the part II.). In particular, we study how to find the value $a^{\frac{b+1}{2}}$ where b is the maximal integer such that $a^b \leq y$. Let us note that this part consists of many technical arguments.

We know that every natural number b can be written in base 2, i.e. there is an integer $k \geq 0$ such that

$$b = \sum_{i=0}^{k} b_i 2^i = b_k 2^k + b_{k-1} 2^{k-1} + \ldots + b_1 2 + b_0,$$

where $b_i \in \{0, 1\}$, for all $i \in \{0, \ldots, k-1\}$, and $b_k = 1$. Therefore, we can express the value a^b as follows

$$a^b = a^{2^k + b_{k-1} 2^{k-1} + \ldots + b_0 2^0} = \prod_{i=0}^{k} a^{b_i 2^i}.$$

The natural number b is such that

$$a^b \leq y < a^{b+1}.$$

According to the discussion above, we can modify this inequalities as

$$a^{2^k} \cdot a^{b_{k-1}2^{k-1}} \cdot \ldots \cdot a^{b_0 2^0} \leq y < \left(a^{2^k} \cdot a^{b_{k-1}2^{k-1}} \cdot \ldots \cdot a^{b_0 2^0}\right) \cdot a. \tag{3.5}$$

Consequently, we have the following inequalities

$$a^{b_{k-1}2^{k-1}} \cdot \ldots \cdot a^{b_0 2^0} \leq \frac{y}{a^{2^k}} < \left(a^{b_{k-1}2^{k-1}} \cdot \ldots \cdot a^{b_0 2^0}\right) \cdot a.$$

If we set $b' = \frac{b}{a^{2^k}}$ and $u = \frac{y}{a^{2^k}}$ then

$$a^{b'} \leq u < a^{b'} \cdot a.$$

As a result, the value b' is the maximal integer such that $a^{b'} \leq u$. We can now employ the same reasoning on the values b' and u. Let k' be maximal such that $k' < k$ and $b_{k'} \neq 0$. Then in follows that

$$b' = \sum_{i=0}^{k-1} b_i 2^i = \sum_{i=0}^{k'} b_i 2^i.$$

From here we see that after at most k repetitions of the procedure, we are in the situation

$$a^{b_0 2^0} \leq \frac{y}{a^{2^k} \cdot \ldots \cdot a^{b_1 2^1}} < a^{b_0 2^0} \cdot a.$$

We know that $a^{b_0 2^0} = a^{b_0}$ and $a^{b_0} \leq a$. Therefore, we have

$$a^{b_0} \leq \frac{y}{a^{2^k} \cdot \ldots \cdot a^{b_1 2^1}} < a^2. \tag{3.6}$$

Let us denote $z' = \prod_{i=1}^{k} a^{b_i 2^{i-1}}$. Then we can observe that

$$z' = \begin{cases} a^{\frac{b}{2}}, & \text{if } b \text{ is even (i.e. } b_0 = 0), \\ a^{\frac{b}{2}-\frac{1}{2}}, & \text{if } b \text{ is odd (i.e. } b_0 = 1). \end{cases}$$

This implies that

$$a^{\frac{b+1}{2}} = \begin{cases} z' \cdot a^{\frac{1}{2}}, & \text{if } b \text{ is even (i.e. } b_0 = 0), \\ z' \cdot a, & \text{if } b \text{ is odd (i.e. } b_0 = 1). \end{cases}$$

Note that the inequalities (3.6) are equivalent to

$$1 \leq \frac{y}{(z')^2} < a, \quad \text{if } b_0 = 0,$$
$$a \leq \frac{y}{(z')^2} < a^2, \quad \text{if } b_0 = 1.$$

We can conclude that the value z from Lemma 3.12 can be obtained as

$$z = \begin{cases} z' \cdot \sqrt{a}, & \text{if } \; 1 \leq \frac{y}{(z')^2} < a, \\ z' \cdot a, & \text{if } \; a \leq \frac{y}{(z')^2} < a^2. \end{cases}$$

Note that this explains why we wanted to construct not only the constant a but also the constant $c = \sqrt{a}$ in the part I.).

Let us summarize the problem which we are trying to solve in this part. We are given values a, $\sqrt{a}$ and y such that $a \leq y$. We want to find the value $a^{\frac{b+1}{2}}$, where b is the maximal integer such that $a^b \leq y$. According to the discussion above, we need to do the following:

1. set $u \leftarrow y$ and $z \leftarrow 1$;
2. find the maximal integer $k \geq 0$ such that $a^{2^k} \leq u$;
3. if $k = 0$ and $u < a$ then output $z \cdot \sqrt{a}$ and halt;
4. if $k = 0$ and $u \geq a$ then output $z \cdot a$ and halt;
5. reassign $u \leftarrow \frac{u}{a^{2^k}}$ and $z \leftarrow z \cdot a^{2^{k-1}}$;
6. goto step 2.

The remaining problem which we need to solve is the step 2. We will study this non-trivial task separately as an auxiliary algorithm.

For a, u such that $1 \leq a \leq u$ we want to find the maximal integer $k \geq 0$ having the property $a^{2^k} \leq u$ (the condition $a \leq u$ ensures that such k always exists). Note that we do not need to know the value k but, in fact, only the values a^{2^k} and $a^{2^{k-1}}$. Let us therefore define two auxiliary values s_{-1}, s_0 as follows

$$s_0 = a^{2^k} \quad \text{and} \quad s_{-1} = \begin{cases} 0, & \text{if } k = 0; \\ a^{2^{k-1}}, & \text{if } k \geq 1. \end{cases} \tag{3.7}$$

The following algorithm shows how to find the values s_{-1}, s_0 given values a and u.

```
Input: $a, u$ such that $1 < a \leq u$
Output: $s_{-1}, s_0$ satisfying equalities (3.7).
$s_{-1} \leftarrow 0$;
$s_0 \leftarrow a$;
$s_1 \leftarrow a^2$;
while $s_1 \leq u$ do
    $s_{-1} \leftarrow s_0$;
    $s_0 \leftarrow s_1$;
    $s_1 \leftarrow s_1^2$;
end
return $s_{-1}, s_0$
```

Algorithm 3.4: Find the maximal integer k such that $a^{2^k} \leq u$

Theorem 3.13. *Given the values u, a such that $1 < a \leq u$, Algorithm 3.4 solves the task of finding values s_{-1}, s_0 which satisfy the equalities (3.7). The time complexity of Algorithm 3.4 is $\mathcal{O}(\log_2(\log_a(u)))$ and the set of used constants is $\{0, 1\}$.*

Proof. For $m \in \mathbb{N}$ let us denote $s_1^{(m)}, s_0^{(m)}, s_{-1}^{(m)}$ the values s_1, s_0, s_{-1} before the m-th while-loop.

We first prove the following auxiliary statement. For every $m \in \mathbb{N}$

$$s_1^{(m)} = a^{2^m}.$$

We proceed by induction on m. After the initialisation we have

$$s_1^{(1)} = a^2 = a^{2^1},$$

thus, the auxiliary statement holds for $m = 1$. Assume now that $s_1^{(m-1)} = a^{2^{m-1}}$. Then we have the following equalities

$$s_1^{(m)} = \left(s_1^{(m-1)}\right)^2 = \left(a^{2^{m-1}}\right)^2 = a^{2^m}.$$

Hence, we proved the auxiliary statement.

It follows that if the algorithm stops then $s_1^{(m)}$ is of the form a^{2^m}, where m equals the number of performed while-loops minus one. In other words, if we set $k = m - 1$, then k is the maximal integer such that $a^{2^k} \leq u$. We derived that the value k from the definition (3.7) equals the number of performed while-loops. Observe that

$$a^{2^k} \leq u \;\Leftrightarrow\; 2^k \leq \log_a(u) \;\Leftrightarrow\; k \leq \log_2\left(\log_a(u)\right). \tag{3.8}$$

We can conclude that the algorithm always terminates.

By induction on k we now prove that the output is correct, i.e. that it satisfies the equalities (3.7). For $k = 0$ we have $s_0 = a^{2^0}$ and $s_{-1} = 0$ as required. For $k \geq 1$ we see that

$$\begin{aligned} s_0^{(k+1)} &= s_1^{(k)} = a^{2^k}, \\ s_{-1}^{(k+1)} &= s_0^{(k)} = s_1^{(k-1)} = a^{2^{k-1}}. \end{aligned}$$

Thus, the algorithm always returns the correct output.

It remains to prove the stated time complexity. As we already discussed, the number of while-loop is less or equal to $\lceil \log_2\left(\log_a(u)\right)\rceil$. There are only three elementary instructions in the initialization phase. The same number of elementary operations is performed in every while-loop. The branch condition needs one elementary instruction. Thus, we can conclude that the time complexity of the algorithm is $\mathcal{O}(\log_2\left(\log_a(u)\right))$.

We see directly from the algorithm that the only used constants are 0 and 1 as stated. $\square$

Now we are finally prepared state Algorithm 3.5 which finds an approximate zero of the polynomial $f(z) = z^2 - y$, where $y \geq 1 + 4\alpha_0$ is the input of the algorithm.

Theorem 3.14. *Algorithm 3.5 succeeds in finding a value z such that*

$$y \leq z^2 \leq y \cdot (1 + 4\alpha_0),$$

where $y > 1 + 4\alpha_0$ is the input of the algorithm. The time complexity of Algorithm 3.5 is $\mathcal{O}\left((\log_2 \log_a(y))^2\right)$ and the set of used constants is $\{0, 1\}$.

The algorithm was derived from the discussion after Lemma 3.12. The fact that the algorithm works follows from this discussion. Therefore, we do not repeat all arguments again and only sketch the proof of the theorem.

```
Input: $y \in \mathbb{R}$ such that $y > 1 + 4\alpha_0$
Output: $z \in \mathbb{R}$ such that $y \leq z^2 \leq y \cdot (1 + 4\alpha_0)$
$(c, a) \leftarrow$ Algorithm3.3();
$u \leftarrow y$;
$z \leftarrow 1$;
$(s_{-1}, s_0) \leftarrow$ Algorithm3.4$(a, u)$;
while $s_{-1} \neq 0$ do
    $u \leftarrow \frac{u}{s_0}$;
    $z \leftarrow z \cdot s_{-1}$;
    if $u < a$ then
        $s_{-1} = 0$;
    end
    else
        $(s_{-1}, s_0) \leftarrow$ Algorithm3.4$(a, u)$;
    end
end
if $u < a$ then
    $z \leftarrow z \cdot c$;
end
else
    $z \leftarrow z \cdot a$;
end
return $z$
```

Algorithm 3.5: Find an approximate zero of the polynomial $z^2 - y \in \mathbb{R}[z]$

Proof (Sketch). The correctness of the result follows from the discussion after Lemma 3.12 and the correctness of Algorithm 3.4, which was proved in Theorem 3.13.

Let k be the maximal integer such that $a^{2^k} \leq y$. From the discussion we also know that after at most k calls to Algorithm 3.4 we are in the situation

$$1 \leq u < a^2.$$

Thus, either $u < a$ and s_{-1} is set to 0. Or $a \leq u < a^2$ and the next call to Algorithm 3.4 returns $s_{-1} = 0$. In both cases $s_{-1} = 0$ which means that the while condition is not satisfied. We derived that Algorithm 3.5 stops after at most k while-loops.

In order to prove the stated time complexity, recall that Algorithm 3.5 performs only constantly many elementary operations. From Theorem 3.13 we know that the time complexity of Algorithm 3.4 is $\mathcal{O}\left(\log_2\left(\log_a(y)\right)\right)$. Thus, for the initialization phase we need $\mathcal{O}\left(\log_2\left(\log_a(y)\right)\right)$ elementary instructions. For every while-loop we need two branch nodes, we perform only constantly many elementary instructions and we call Algorithm 3.4 once. Algorithm 3.4 is called with input value $u < y$. Therefore, the time complexity of every while-loop is $\mathcal{O}\left(\log_2\left(\log_a(y)\right)\right)$. We know that there are at most k while-loops. Recall the inequality (3.8), which tells us that

$$k \leq \log_2\left(\log_a(y)\right).$$

Before we output the result we branch once and then perform one elementary instruction. All together, the time complexity of Algorithm 3.5 is

$$\mathcal{O}\left(\log_2\left(\log_a(y)\right)\right) + \log_2\left(\log_a(y)\right) \cdot \mathcal{O}\left(\log_2\left(\log_a(y)\right)\right).$$

The expression can be simplified to

$$\mathcal{O}\left(\left(\log_2\left(\log_a(y)\right)\right)^2\right).$$

Finally, observe that Algorithm 3.5 uses only constants from the set $\{0, 1\}$ and constants that are used in Algorithm 3.3 and Algorithm 3.4. But we have already discussed that the only constants appearing in these auxiliary algorithms are 0 and 1. $\square$

Now we are prepared to define the final algorithm that approximates the square root of a given non-negative real number y with the precision $\varrho > 0$.

Theorem 3.15. *Algorithm 3.6 given an input $y \geq 1 + 4\alpha_0$ returns a value z such that*

$$|z^2 - y| < \varrho.$$

The time complexity of the algorithm is $\mathcal{O}\left((\log_2(\log_a(y) - \log_a(\varrho)))^2\right)$ while the set of used constants is $\{0, 1\}$.

Proof. The correctness of our algorithm follows from the correctness of Newton's method. From Theorem 3.14 we know that Algorithm 3.5 returns an approximate zero of the polynomial $f(z) = z^2 - y$. Thus, Newton's algorithm converges by Lemma 3.9. This proves that Algorithm 3.6 halts on every input.

```
Input: $y \in \mathbb{R}$ such that $y \geq 1 + 4\alpha_0$
Output: $z \in \mathbb{R}$ such that $|z^2 - y| < \varrho$
$z_0 \leftarrow$ Algorithm3.5($y$);
$z \leftarrow \frac{1}{2} \cdot (z_0 + \frac{y}{z_0})$;
while $|z^2 - y| \geq \varrho$ do
    $z \leftarrow \frac{1}{2} \cdot (z + \frac{y}{z})$;
end
return $z$
```

Algorithm 3.6: Approximation of $\sqrt{y}$ based on Newton's method

Let us now discuss its time complexity. From Theorem 3.14 we know that the time complexity of finding an approximate zero z_0 is

$$\mathcal{O}\left((\log_2(\log_a(y)))^2\right).$$

Lemma 3.9 tell us that for $0 < \varrho < 1$ we need $\mathcal{O}\left(\log_2(\log_2\left(\frac{L}{\varrho}\right))\right)$ iterations of the Newton endomorphism, where L is the upper bound on the distance between the approximate zero and its associated zero. In other words, L is such that

$$|z_0 - \sqrt{y}| \leq L.$$

From Theorem 3.10 we know that

$$|z_0 - \sqrt{y}| \leq 2\beta(f, z_0).$$

Recall that the quantity $\beta(f, z_0)$ for the polynomial $f(z) = z^2 - y$ is

$$\beta(f, z_0) = \left|\frac{z_0^2 - y}{2z_0}\right|.$$

The equalities (3.4) imply that the output of Algorithm 3.5 is such that

$$y \leq z_0^2 \leq y(1 + 4\alpha_0),$$

which is equivalent to

$$0 \leq z_0^2 - y \leq 4\alpha_0 y.$$

Since $y > 1 + 4\alpha_0$, we have

$$y \leq z_0^2 \Rightarrow 1 < z_0^2 \Rightarrow 1 < z_0.$$

Also recall that $0 < \alpha_0 < \frac{1}{4}$. Putting all these observations together we can conclude that

$$2\beta(f, z_0) = \left|\frac{z_0^2 - y}{z_0}\right| \leq |z_0^2 - y| = z_0^2 - y \leq 4\alpha_0 y \leq y.$$

Thus, we need $\mathcal{O}\left(\log_2\left(\log_2\left(\frac{y}{\varrho}\right)\right)\right)$ iterations of the Newton endomorphism in order to compute the value z given the approximate zero z_0. We see that the computation of the Newton endomorphism for $f(z) = z^2 - y$ is done by three elementary operations. Hence, we can find z knowing z_0 in time $\mathcal{O}\left(\log_2\left(\log_2\left(\frac{y}{\varrho}\right)\right)\right)$.

Note that for $0 < \varrho < 1$ we have

$$\log_2\left(\log_a(y)\right) \leq \log_2\left(\log_a\left(\frac{y}{\varrho}\right)\right).$$

Since $1 < a < 2$, we also know that

$$\log_2\left(\log_2\left(\frac{y}{\varrho}\right)\right) \leq \log_2\left(\log_a\left(\frac{y}{\varrho}\right)\right).$$

The stated time complexity of Algorithm 3.6 follows.

It remains to prove that the set of used constants is $\{0, 1\}$. We know from Theorem 3.14 that, in order to find the value z_0, we need only 0 and 1. In the second part of the Algorithm 3.6 we use only $\{0, 1\}$ since the constant $\frac{1}{2}$ can be computed from the constant 1 by two elementary operations. $\square$

Let us summarize this subsection. We presented two algorithms both of which approximate the square root of a non-negative integer with an arbitrary but fixed precision ϱ. The second introduced algorithm is more efficient than the first one. It is based on Newton's method and we proved that its time complexity is $\mathcal{O}\left((\log_2(\log_a(y) - \log_a(\varrho)))^2\right)$, where $a \approx 1.6$. It is natural to ask whether there is an algorithm approximating the square root function whose time complexity is even better. We study this question in the following subsection.

3.3.3 Lower bound on the time complexity

In this subsection we discuss a lower bound on the time complexity of computing the square root function by a BSS machine. Without lost of generality, we can restrict ourselves to elementary BSS machines. See Observation 2.20 in Section 2.2 for more details. The number of elementary instructions performed by an elementary BSS machine M on an input $y \in \Omega_M$ is strictly smaller than the halting time $T_M(y)$. Therefore, it is sufficient to find a lower bound on the number of elementary instructions.

Consider a rational function φ computed by an elementary BSS machine M in $k \in \mathbb{N}$ elementary instructions. As usual, $\mathbf{C}$ denotes the set of machine constants. Recall what we know from Section 2.2. By Theorem 2.23 the rational function φ is represented by a fraction of two fixed polynomials $p, q \in \mathbb{R}[y]$ such that

$$\varphi = \frac{p}{q}, \tag{3.9}$$

$$\deg(p), \deg(q) \leq 2^k, \tag{3.10}$$

$$\mathbf{coef}(p), \mathbf{coef}(q) \subseteq 2^{2^{k+1}} \cdot \left(\widetilde{\mathbf{C}}^{2^k}\right), \tag{3.11}$$

where $\widetilde{\mathbf{C}} = \mathbf{C} \cup -\mathbf{C}$. Thus, by Lemma 1.13 we have

$$\deg(p^2), \deg(q^2) \leq 2^{k+1},$$

$$\mathbf{coef}(p^2), \mathbf{coef}(q^2) \subseteq 2^k \cdot \left(2^{2^{k+1}} \cdot \left(\widetilde{\mathbf{C}}^{2^k}\right)\right)^2.$$

The expression on the right hand side of the inclusion can be simplified using relations stated in Lemma 1.3 as

$$2^{2^k}\left(2^{2^{k+1}}\cdot\left(\widetilde{\mathbf{C}}^{2^k}\right)\right)^2 = 2^{2^{k+2}+2^k}\cdot\left(\widetilde{\mathbf{C}}^{2^{k+1}}\right)\subseteq 2^{2^{k+3}}\cdot\left(\widetilde{\mathbf{C}}^{2^{k+1}}\right).$$

The inclusion follows from the fact that $2^{k+2}+2^k \le 2^{k+3}$ for any $k\in\mathbb{N}$. Let us recall the following notation. For $\mathbf{A}\subseteq\mathbb{R}$ we defined

$$\begin{aligned}
&\max_{\mathbf{A}} = \max\{|c|\colon c\in\mathbf{A}\},\\
&\min_{\mathbf{A}} = \min\{|c|\colon c\in\mathbf{A}\setminus\{0\}\},\\
&\mu(t,\mathbf{A}) = \min_{t\mathbf{A}}.
\end{aligned}$$

For a polynomial p we will write $\max_p$ (respectively, $\min_p$) instead of $\max_{\mathbf{coef}(p)}$ (respectively, $\min_{\mathbf{coef}(p)}$). We believe that this simplified notation will not cause any confusion.

Furthermore, from Observation 1.6 we know that $\max_{\mathbf{C}} = \max_{-\mathbf{C}}$. Consequently, $\max_{\mathbf{C}} = \max_{\widetilde{\mathbf{C}}}$. Using the simplified notation we can conclude that

$$\max_{p^2},\max_{q^2} \le 2^{2^{k+3}}\cdot\left(\max_{\mathbf{C}}\right)^{2^{k+1}}, \tag{3.12}$$

$$\min_{p^2},\min_{q^2} \ge \mu\left(2^{2^{k+3}},\widetilde{\mathbf{C}}^{2^{k+1}}\right). \tag{3.13}$$

Before we return back to our original task, let us make one general observation, which follows from the triangular inequality.

Observation 3.16. *Let $p,q\in\mathbb{R}[y]$ be two polynomials and let $r\in\mathbb{R}[y]$ be such that $r=p-q$. Then $\max_r \le \max_p + \max_q$.*

Now we are prepared to state the main theorem of this passage. It will be convenient to view M as a machine that inverts the square function rather than a machine that computes the square root function. The relationship between these to approaches was discussed in detail in Section 3.1.

Theorem 3.17. *Let $\epsilon>0$. Let M be an elementary BSS machine inverting the square function with the precision ϵ. Let $\mathbf{C}$ be the set of machine constants. Consider $\gamma\in\Gamma_M$ to be a halting path with $k\in\mathbb{N}$ computation nodes (elementary instructions). Let φ be the corresponding rational function and let $p,q\in\mathbb{R}[y]$ be the fixed polynomials whose fraction represents the rational function φ. Let $\nu\subset\Omega_M$ be the semi-algebraic set corresponding to the halting path γ. Then the following inclusion holds*

$$\left\{y\in\nu\colon\ \left|\varphi^2(y)-y\right|<\epsilon\right\}\subseteq\left[0,1+\frac{\max_{p^2}+\max_{q^2}+\epsilon\cdot\max_{q^2}}{\min\{\min_{p^2},\min_{q^2}\}}\right].$$

Let us first make two remarks. We know that $\Omega_M=\mathbb{R}^{\ge 0}$ must be the case since we assume M to be a machine that inverts the square function whose range is $\mathbb{R}^{\ge 0}$. Next, recall from the beginning of this subsection that the polynomials p,q from the statement satisfy the conditions (3.9) - (3.11).

Before we formally prove the theorem, let us explain the main idea of the proof. Suppose that we have a polynomial $\psi\in\mathbb{R}[y]$ whose leading coefficient is positive and for which there exists $n\in\mathbb{R}^{\ge 0}$ such that

$$\left\{y\in\nu\colon\ \left|\varphi^2(y)-y\right|<\epsilon\right\}\subseteq[0,n]\cup\{y\in\nu\colon\psi(y)<0\}.$$

Then we can apply Lemma 1.16 on the polynomial ψ and obtain a bound n_ψ on the absolute value of the zeros of the polynomial ψ. From Corollary 1.17 we know that n_ψ is such that

$$\{y \in \mathbb{R}^{\geq 0} \colon \psi(y) < 0\} \subseteq [0, n_\psi].$$

Thus, for $N = \max\{n, n_\psi\}$ we would have

$$\{y \in \nu \colon \left|\varphi^2(y) - y\right| < \epsilon\} \subseteq [0, N].$$

In the proof we show that the bound N can be chosen as stated in the theorem.

Let us make one final remark before we proceed to the proof. Since $\varphi^2 = \frac{p^2}{q^2}$ and the rational function is defined for every $y \in \nu$, the following two sets are the same

$$\{y \in \nu \colon \left|\varphi^2(y) - y\right| < \epsilon\} = \{y \in \nu \colon |p^2(y) - yq^2(y)| < \epsilon q^2(y)\}.$$

Proof of Theorem 3.17. We will proceed by a case analysis splitting on the degree of polynomials p and q.

(I) Assume first that $\deg(p) = \deg(q) = 0$.
In this case $p(y) = \alpha, q(y) = \beta \neq 0$ for some $\alpha, \beta \in \mathbb{R}$. Then for every $y \geq \frac{\alpha^2}{\beta^2}$ we have

$$|\alpha^2 - y\beta^2| < \epsilon\beta^2 \;\Leftrightarrow\; y\beta^2 - \alpha^2 < \epsilon\beta^2 \;\Leftrightarrow y < \frac{\alpha^2}{\beta^2} + \epsilon.$$

Consequently, we achieve the following inclusion

$$\{y \in \nu \colon \left|\varphi^2(y) - y\right| < \epsilon\} \subseteq \left[0, \frac{\alpha^2}{\beta^2} + \epsilon\right] \subseteq \left[0, \frac{\max_{p^2}}{\min_{q^2}} + \epsilon\right].$$

(II) Suppose now that $\deg(p) \neq 0$ or $\deg(q) \neq 0$.

(1) We first discuss the situation when $\deg(p) > \deg(q)$. In this case we know the leading coefficient of the polynomial $p^2 - yq^2$, i.e.

$$\operatorname{lc}\left(p^2 - yq^2\right) = \operatorname{lc}(p^2) > 0.$$

By Observation 3.16 and the fact that $\mathbf{coef}(q^2) = \mathbf{coef}(y \cdot q^2)$ we have

$$\max_{p^2 - yq^2} \leq \max_{p^2} + \max_{q^2}.$$

Now we apply Lemma 1.16 on the polynomial $p^2 - yq^2$ and obtain the following value

$$n = 1 + \frac{\max_{p^2} + \max_{q^2}}{\min_{p^2}},$$

which bounds the absolute value of zeros of the polynomial $p^2 - yq^2$. Since $\operatorname{lc}(p^2 - yq^2) > 0$, by Corollary 1.17 we can conclude that for every $y > n$ the following inequality holds

$$p^2 - yq^2 > 0.$$

Consequently, for $y > n$ we have the equivalence

$$|p^2(y) - yq^2(y)| < \epsilon q^2(y) \ \Leftrightarrow p^2(y) - yq^2(y) - \epsilon q^2(y) < 0.$$

Let us now define a polynomial ψ as

$$\psi(y) = p^2(y) - yq^2(y) - \epsilon q^2(y).$$

Recall that we study the case where $\deg(p) > \deg(q)$; therefore, we have $\mathrm{lc}(\psi) = \mathrm{lc}(p^2)$. Lemma 1.16 together with Observation 3.16 gives us the bound

$$n_\psi = 1 + \frac{\max_{p^2} + \max_{q^2} + \epsilon \cdot \max_{q^2}}{\min_{p^2}}.$$

Because the leading coefficient of ψ is positive, the bound derived above is such that for every $y > n_\psi$ the value $\psi(y)$ is positive (see Corollary 1.17). Moreover, note that $n < n_\psi$. In conclusion,

$$\left\{y \in \nu \colon \left|\varphi^2(y) - y\right| < \epsilon\right\} \subseteq \left[0, 1 + \frac{\max_{p^2} + \max_{q^2} + \epsilon \cdot \max_{q^2}}{\min_{p^2}}\right].$$

(2) Let us focus on the remaining case, i.e. when $\deg(p) \leq \deg(q)$. Then

$$\mathrm{lc}\left(p^2(y) - yq^2(y)\right) = -\mathrm{lc}(q^2(y)) < 0.$$

From Lemma 1.16 and Observation 3.16 we obtain the following bound

$$n = 1 + \frac{\max_{p^2} + \max_{q^2}}{\min_{q^2}}.$$

From Corollary 1.17 we know that the bound is of the following property. For all $y > n$ it holds that

$$p^2(y) - yq^2(y) < 0.$$

Hence, for $y > n$ we have

$$|p^2(y) - yq^2(y)| < \epsilon q^2(y) \ \Leftrightarrow yq^2(y) - p^2(y) - \epsilon q^2(y) < 0.$$

Now we can set $\psi(y) = yq^2(y) - p^2(y) - \epsilon q^2(y)$. In this case the leading coefficient is $\mathrm{lc}(\psi) = \mathrm{lc}(q^2)$ which is positive. Thus, we can use Lemma 1.16 on the polynomial ψ to obtain a bound n_ψ such that $\psi(y) > 0$ for every $y > n_\psi$. By Observation 3.16 we have

$$n_\psi = 1 + \frac{\max_{p^2} + \max_{q^2} + \epsilon \cdot \max_{q^2}}{\min_{q^2}}.$$

Finally, we can conclude that

$$\left\{y \in \nu \colon \left|\varphi^2 - y\right| < \epsilon\right\} \subseteq \left[0, 1 + \frac{\max_{p^2} + \max_{q^2} + \epsilon \cdot \max_{q^2}}{\min_{q^2}}\right].$$

□

From Theorem 3.17 we can derive a lower bound on the number of elementary operations which are needed in order to invert x^2 with an arbitrary but fixed precision $\epsilon > 0$. Let us discuss it in more detail.

Let $\epsilon > 0$ be a fixed constant and consider the interval $[-\sqrt{N}, \sqrt{N}]$ for a natural number $N \in \mathbb{N}$. Suppose that M is an elementary BSS machine such that for every $x \in [-\sqrt{N}, \sqrt{N}]$ it holds that

$$|(\Phi_M(x^2))^2 - x^2| < \epsilon.$$

In other words, the machine M is such that

$$\{y \in \Omega_M \colon |(\Phi_M(y))^2 - y| < \epsilon\} \supseteq [0, N].$$

Then in particular

$$\{y \in \Omega_M \colon |(\Phi_M(y))^2 - y| < \epsilon\} \not\subseteq [0, N-1].$$

Thus, by Theorem 3.17 there must be a halting path $\gamma \in \Gamma_M$ such that for the corresponding rational function φ, which is represented by the fixed pair of polynomials $p, q \in \mathbb{R}[y]$, we have the following inequality

$$N - 1 < 1 + \frac{\max_{p^2} + \max_{q^2} + \epsilon \cdot \max_{q^2}}{\min\{\min_{p^2}, \min_{q^2}\}}.$$

The inequality can be equivalently expressed as

$$N \leq 1 + \frac{\max_{p^2} + \max_{q^2} + \epsilon \cdot \max_{q^2}}{\min\{\min_{p^2}, \min_{q^2}\}}. \tag{3.14}$$

The values $\max_{p^2}, \max_{q^2}, \min_{p^2}, \min_{q^2}$ depend on the set of machine constants $\mathbf{C}$ and the number of elementary instructions k of the machine M. We can express the inequality (3.14) in terms of the bounds (3.12), (3.13) which were derived at the beginning of this section. We achieve

$$N \leq 1 + \frac{2^{2^{k+3}}(2+\epsilon) \cdot (\max(\mathbf{C}))^{2^{k+1}}}{\mu\left(2^{2^{k+3}}, \widetilde{\mathbf{C}}^{2^{k+1}}\right)}.$$

To derive a general lower bound on the number of elementary instructions of a machine inverting x^2, we would need to express the value k from the formula above. Unfortunately, as we already discussed in the Section 2.2, finding the value $\mu\left(2^{2^{k+3}}, \widetilde{\mathbf{C}}^{2^{k+1}}\right)$ for a general set $\mathbf{C}$ is probably very difficult. On the other hand, for many sets it is easy. Thus, in concrete instances we are able to express a concrete lower bound. Let us discuss one simple but very useful and realistic example.

Lower bound in the special case $\mathbf{C} = \{0, 1\}$

There is a good reason for studying this special case of the general analysis from above. Consider an adversary which is trying to invert the input-output map of some machine M_0. It is reasonable to assume that the adversary can use (only) the same constants as the machine M_0. In this section we are focusing on inverting the square function with some fixed precision $\epsilon > 0$. Thus, the machine M_0 is a very simple machine that computes the square function. Note that there are no constants needed. For technical reasons we assume that $\mathbf{C}_{M_0} = \{0, 1\}$. Hence, the adversary has to construct a machine M which inverts the square function with the precision $\epsilon > 0$ and uses only constant 0 and 1, i.e. $\mathbf{C}_M = \{0, 1\}$.

Let us first state an auxiliary lemma about the set $\mathbf{C} = \{0, 1\}$. Note that $\widetilde{\mathbf{C}} = \{-1, 0, 1\}$.

Lemma 3.18. *Let $\mathbf{C} = \{0, 1\}$, $t \in \mathbb{N}$, $s \in \mathbb{N}$. Then the sets $t \cdot \mathbf{C}^s$ and $t \cdot \widetilde{\mathbf{C}}^s$ have the same minimal and maximal size of their elements, in particular*

$$\max_{t \cdot \mathbf{C}^s} = \max_{t \cdot \widetilde{\mathbf{C}}^s} = t,$$
$$\min_{t \cdot \mathbf{C}^s} = \min_{t \cdot \widetilde{\mathbf{C}}^s} = 1.$$

Proof. From Observation 1.7 and the fact that $\max_{\mathbf{C}} = \max_{\widetilde{\mathbf{C}}} = 1$ we have

$$\max_{t \cdot \mathbf{C}^s} = t \cdot (\max_{\mathbf{C}})^s = t \cdot (1)^s = t \cdot \left(\max_{\widetilde{\mathbf{C}}}\right)^s = \max_{t \cdot \widetilde{\mathbf{C}}^s}.$$

From Observation 1.7 and the fact that $\min_{\mathbf{C}} = \min_{\widetilde{\mathbf{C}}} = 1$ we have

$$\min_{\mathbf{C}^s} = \min_{\mathbf{C}} = 1 = \min_{\widetilde{\mathbf{C}}} = \min_{\widetilde{\mathbf{C}}^s}.$$

Since there is no negative element in the set $\mathbf{C}^s$, we can conclude that

$$\min_{t \cdot \mathbf{C}^s} = \min_{\mathbf{C}^s} = 1.$$

In addition,

$$\underbrace{\{-1, 0, 1\} + \cdots + \{-1, 0, 1\}}_{t} = \{-t, \ldots, -1, 0, 1, \ldots, t\}.$$

From here we see that the element of the minimal size of the set $t\widetilde{\mathbf{C}}^s$ is again 1. $\square$

Our aim is to find a lower bound on the number of elementary operations which are necessary for a successful inversion of the square function. Let p, q be polynomials defined as in the general case. Now we can express the general formulae (3.12), (3.13) explicitly. We achieve the following bounds

$$\max_{p^2}, \max_{q^2} \leq 2^{2^{k+3}} \cdot (\max_{\mathbf{C}})^{2^{k+1}} = 2^{2^{k+3}},$$
$$\min_{p^2}, \min_{q^2} \geq \mu\left(2^{2^{k+3}}, \widetilde{\mathbf{C}}^{2^{k+1}}\right) = 1,$$

where k is the number of elementary operations of the machine M. We can use this results in the inequality (3.14) and achieve

$$N \leq 1 + \frac{\max_{p^2} + \max_{q^2} + \epsilon \cdot \max_{q^2}}{\min\{\min_{p^2}, \min_{q^2}\}} \leq 1 + \frac{2^{2^{k+3}} + 2^{2^{k+3}} + \epsilon \cdot 2^{2^{k+3}}}{1}.$$

From here we can derive the following lower bound on the value k

$$\frac{N-1}{2+\epsilon} \leq 2^{2^{k+3}} \Leftrightarrow \log_2\left(\log_2\left(\frac{N-1}{2+\epsilon}\right)\right) - 3 \leq k.$$

Let us conclude the discussion in the following corollary.

Corollary 3.19. *Any elementary BSS machine inverting x^2 with the precision $\epsilon > 0$ for $x \in [-\sqrt{N}, \sqrt{N}]$ has to either make $\Omega(\log_2(\log_2(N) - \log_2(\epsilon)))$ elementary instructions or use more constants than just 0 and 1.*

3.3.4 Summary

Let us recapitulate what we proved about BSS machines that invert the square function with the precision $\epsilon > 0$.

We first showed that there is no constant time BSS machine that succeeds on every real number.

Secondly, we presented two BSS machines which approximate the square root function with the precision $\varrho > 0$ and use only 0 and 1 as machine constants. The first machine is based on the half-interval search algorithm and succeeds on any positive real input. If the input y is from the interval $[0, 1)$ then the time complexity of the machine is constant. If the input y is greater or equal to 1 then the time complexity is $\mathcal{O}(\log_2(y) - \log_2(\varrho))$. The second machine succeeds on inputs y which are greater or equal to $a \approx 1.5625$. Nevertheless, its time complexity is slightly better; namely, $\mathcal{O}\left((\log_2(\log_a(y) - \log_a(\varrho)))^2\right)$. Recall from Section 3.1 that in order to *invert the square function* with the precision ϵ, we have to *approximate the square root function* with the precision $0 < \varrho \leq \sqrt{y + \epsilon} - \sqrt{y}$. Thus, we proved that the time complexity of inverting the square function with the precision ϵ is

$$\mathcal{O}\left(\left(\log_2\left(\log_a(x^2) - \log_a(\sqrt{x^2 + \epsilon} - x)\right)\right)^2\right),$$

where $a \approx 1.5625$.

And finally, we showed that no BSS machine can invert the square function with the precision $\epsilon > 0$ on the interval $[-\sqrt{N}, \sqrt{N}]$ in less steps than

$$\Omega\left(\log_2\left(\log_2(N) - \log_2(\epsilon)\right)\right),$$

while using only constants 0 and 1. In other words, we have a lower bound of the form: either the number of steps is $\Omega(\log_2(\log_2(N) - \log_2(\epsilon)))$ or the set of machine constants is larger than $\{0, 1\}$.

4. Inverting a constant time BSS machine over $\mathbb{R}$

Imagine the following situation. We have a real function $\psi\colon \mathbb{R} \to \mathbb{R}$ which is computable by a BSS machine M_0. The machine M_0 is publicly known. We run the machine M_0 on a secret value $x \in \Omega_{M_0}$ and publish the output $y = \Phi_{M_0}(x)$. The aim of this chapter is to answer the following questions.

Can an adversary who knows the machine M_0 and the value y find $x' \in \Omega_{M_0}$ such that

$$\Phi_{M_0}(x') = \Phi_{M_0}(x)?$$

If the answer is no, can he at least find a value $x' \in \Omega_{M_0}$ such that

$$|\Phi_{M_0}(x') - \Phi_{M_0}(x)| < \epsilon,$$

where $\epsilon > 0$ is a fixed constant? In other words, is he able to construct a BSS machine M that inverts the machine M_0?

In the previous chapter we discussed in detail the situation for the function $\psi(x) = x^2$. In this chapter we analyse the inversion problem for a general real function $\psi\colon \mathbb{R} \to \mathbb{R}$ which is easy to compute by a BSS machine. But let us proceed more systematically and first make the expressions "easy to compute" and "invert a machine" more formal.

4.1 How to define inversion

First of all observe that a BSS machine that computes a real function $\psi\colon \mathbb{R} \to \mathbb{R}$ is a finite dimensional BSS machine with one dimensional input space and one dimensional output space. This is also the reason why finite dimensional BSS machines are of our main interest in this work.

Recall from Section 2.5 that for finite dimensional BSS machines we distinguish the following three situations:

- the function ψ is not computable;
- the function is computable but not in constant time;
- the function is computable in constant time.

It is natural to formalize the expression "easy to compute" as follows.

Definition 4.1. We say that a real function $\psi\colon \mathbb{R} \to \mathbb{R}$ is *easy to compute* if there exists a constant time BSS machine computing it.

Recall from Section 2.2 that for every BSS machine there exists an elementary BSS machine which computes the same function, has the same halting set and its halting time is proportionally the same.

For the rest of this chapter we fix an arbitrary elementary constant time BSS M_0 with one dimensional input space $\mathcal{I}_{M_0} = \mathbb{R}$ and one dimensional output space

$\mathcal{O}_{M_0} = \mathbb{R}$. As we have already said, our aim is to study BSS machines that invert the machine M_0. In order to do so, we need to specify what we mean by "inversion of a BSS machine".

There are more possible ways how to define that a BSS machine M inverts the value $\Phi_{M_0}(x)$ for $x \in \Omega_{M_0}$. Probably the most intuitive approach is to require infinite precision as in the classical Turing model of computation, i.e. for $x \in \Omega_{M_0}$ we say that the machine M inverts the value $\Phi_{M_0}(x)$ if

$$\Phi_M(\Phi_{M_0}(x)) = x.$$

Another possibility, which corresponds better with the nature of real numbers, is to allow the machine M to make a small mistake. In particular, let $\varrho > 0$ be a fixed constant. Then for $x \in \Omega_{M_0}$ we say that the machine M computes the inverse of the value $\Phi_{M_0}(x)$ with the precision ϱ if

$$|\Phi_M(\Phi_{M_0}(x)) - x| < \varrho.$$

The function Φ_{M_0} might not be injective; hence, there exist $x, x' \in \Omega_{M_0}$ such that $x \neq x'$ and $\Phi_{M_0}(x) = \Phi_{M_0}(x')$. The machine M which achieves the input $\Phi_{M_0}(x) = \Phi_{M_0}(x')$ has now way how to decided between x and x'. This motivates the following definition.

Definition 4.2. Let M_0 be a BSS machine with one dimensional input space and one dimensional output space.

1. We say that a BSS machine M *inverts the BSS machine* M_0 *with infinite precision* if
$$\forall x \in \Omega_{M_0} \colon \Phi_{M_0}(\Phi_M(\Phi_{M_0}(x))) = \Phi_{M_0}(x).$$

2. Let $\epsilon > 0$ be a fixed constant. Then we say that a BSS machine M *inverts the BSS machine* M_0 *with the precision* ϵ if
$$\forall x \in \Omega_{M_0} \colon |\Phi_{M_0}(\Phi_M(\Phi_{M_0}(x))) - \Phi_{M_0}(x)| < \epsilon.$$

Sometimes we refer to the first variant as inversion with the precision $\epsilon = 0$.

Remark 4.3. If we say that a BSS machine inverts a function then we mean that a BSS machine inverts a BSS machine which computes the function.

For reasons discussed already in Chapter 3, the first variant of Definition 4.2 is not appropriate while working with real numbers. Hereafter, when we say "invert", we implicitly mean invert with the precision $\epsilon > 0$.

4.2 The main idea of our construction

In this chapter we prove that for every BSS machine M_0 as defined above there exists a BSS machine M inverting the machine M_0. Our prove will be constructive which means that we describe such a machine M explicitly. In addition, we also discuss the time complexity of the constructed machine although it requires many technical computations.

We first discuss the consequences of M_0 being a constant time BSS machine. We also recall important properties of finite dimensional machines and establish the notation for the rest of this chapter.

By definition there exists a natural number $T \in \mathbb{N}$ such that $T_{M_0}(x) \leq T$ for every $x \in \Omega_{M_0}$. From Lemma 2.5 we know that the set of halting paths is bounded, i.e.

$$\Gamma_{M_0} \leq 2^T.$$

By the Path Decomposition Theorem we know that

$$\Omega_{M_0} = \dot{\bigcup_{\gamma \in \Gamma_{M_0}}} \nu_\gamma$$

is a finite disjoint union of basic semi-algebraic sets and that

$$\varphi_\gamma = \Phi_{M_0} \restriction_{\nu_\gamma}$$

is a rational function for every $\gamma \in \Gamma_{M_0}$. Consequently, we have the following observation.

Observation 4.4. *A real function $\psi\colon \mathbb{R} \to \mathbb{R}$ is easy to compute in sense of Definition 4.1 if it is defined by finitely many rational functions.*

By Remark 1.10 every semi-algebraic subset of $\mathbb{R}$ is a finite union of open intervals and points. Thus, by Corollary 2.16 the halting set Ω_{M_0} is a finite disjoint union of open intervals and points

$$\Omega_{M_0} = \dot{\bigcup_{\gamma \in \Gamma_{M_0}}} \left(\dot{\bigcup_{i \in \{1,\ldots,m_\gamma\}}} I_{\gamma,i} \right),$$

where m_γ is the number of connected components of the semi-algebraic set ν_γ.

In addition, we know that for every $\gamma \in \Gamma_{M_0}$ the rational function φ_γ is continuous on $I_{\gamma,i}$, where $i \in \{1, \ldots, m_\gamma\}$.

Let m be the number of connected components of the halting set. Then from Corollary 2.26 we have the following bound

$$m = \sum_{\gamma \in \Gamma_{M_0}} m_\gamma \leq 2^{T+1} \cdot 3^{T-1}.$$

For every $\gamma \in \Gamma_{M_0}$ and every $i \in \{1, \ldots, m_\gamma\}$ we define $J_{\gamma,i}$ as

$$J_{\gamma,i} = \varphi_\gamma(I_{\gamma,i}).$$

Since φ_γ is a rational function which is continuous on $I_{\gamma,i}$, we know that $J_{\gamma,i}$ is a point or an open interval as well.

Consequently, the following equality holds

$$\Phi_{M_0}(\Omega_{M_0}) = \bigcup_{\gamma \in \Gamma_{M_0}} \left(\bigcup_{i \in \{1,\ldots,m_\gamma\}} J_{\gamma,i} \right).$$

Remark 4.5. The components $J_{\gamma,i}$ are generally not disjoint. That happens only when the input-output map of the machine M_0 is injective.

Let us return back to our problem. We have a fixed constant $\epsilon > 0$ and we consider a BSS machine M_0 with the properties described at the beginning of this chapter. Our aim is to show how an adversary constructs a BSS machine M which inverts the machine M_0 with the fixed precision ϵ. According to the discussion above we can divide the construction in two steps:

1. Given $y = \Phi_{M_0}(x)$, find a component $J_{\gamma,i}$ such that $y \in J_{\gamma,i}$.

The knowledge of $J_{\gamma,i}$ determines the corresponding path γ, the interval or point $I_{\gamma,i}$ and the rational function φ_γ. Therefore, the second task is:

2. Find $x' \in I_{\gamma,i}$ such that $|\varphi_\gamma(x') - y| < \epsilon$.

We discuss each part separately. In Section 4.3 we prove that there is a BSS machine solving the first problem and in Section 4.4 we show how to construct a BSS machine that fulfills the second task.

Before be proceed to the next section, let us make one last remark. Every BSS machine has, according to the definition, a finite set of machine constants. This is, indeed, a very powerful property. Since we are inverting a fixed BSS machine M_0, the inverting machine M can have finitely many values describing the machine M_0 as machine constants (for example the endpoints of the connected components $I_{\gamma,i}$ and $J_{\gamma,i}$ or the coefficients of the rational function φ_γ and so on). It is clear that we can store only values that are independent of the value y.

Let us establish the following terminology. If we say that an adversary can "precompute" some value, it means that it is a property of the publicly known machine M_0 which is independent of the value y so it can be considered as a machine constant of the inverting machine M.

4.3 Step 1: Finding an interval

Assume now that we have finitely many open intervals and points $J_1, \dots, J_m$. In this section we show that there is a constant time machine M which, given a value

$$y \in \bigcup_{i=1}^{m} J_i,$$

returns an index $i \in \{1, \dots, m\}$ such that $y \in J_i$.

We are inverting a concrete machine M_0. Therefore, we can assume that an adversary knows the components $J_1, \dots, J_m$. Let us be more precise. Every interval is uniquely determined by its endpoints. Thus, there are at most $2m$ values needed in order to store a complete information about the components $J_1, \dots, J_m$. So the inverting machine M has all these values as machine constants; in particular,

$$c_1, d_1, \dots, c_m, d_m \in \mathbf{C}_M,$$

where c_i, d_i are endpoints of the interval J_i. In case J_i is a point then $c_i = d_i = J_i$.

Consequently, the machine M can search through the components $J_1, \dots, J_m$ exhaustively as described in Algorithm 4.1. Such a machine is certainly not

```
**Input:** $y$ such that $y \in \bigcup_{i=1}^{m} J_i$
**Output:** $i$ such that $y \in J_i$
$i \leftarrow 1$;
found $\leftarrow$ false;
**while** found $=$ false **do**
    **if** $(c_i < y < d_i)$ *or* $((y = c_i)$ *and* $(y = d_i))$ **then**
        found $\leftarrow$ true;
    **end**
    **else**
        $i \leftarrow i + 1$;
    **end**
**end**
**return** $i$
```

Algorithm 4.1: Exhaustive search through intervals

the most efficient machine solving the problem. However, it is sufficient for our purposes.

It is straight forward to prove the following lemma.

Lemma 4.6. *Algorithm 4.1 halts on every valid input $y \in \bigcup_{i=1}^{m} J_i$ and it succeeds in the task of finding an index $i \in \{1, \ldots, m\}$ such that $y \in J_i$. The time complexity of the algorithm is $\mathcal{O}(m)$.*

The halting time of the constructed machine M depends only on the number of components, which is fixed (it does not depend on the input value y). As a consequence, the constructed machine is a constant time BSS machine.

However, its time complexity is exponential in the time complexity of the machine M_0. Recall that $m \leq 2^{T+1} \cdot 3^{T-1}$, where T is the time bound of the machine M_0. Therefore, it is interesting to study a lower bound on the time complexity of the problem of finding an interval J_i such that $y \in J_i$. We briefly do so in the next subsection.

4.3.1 Lower bound on the time complexity

Let us first assume that the components $J_1, \ldots, J_m$ are disjoint open intervals.

Theorem 4.7. *Let $J_1, \ldots, J_m$ be open intervals that are disjoint. Let M be a BSS machine such that given a value $y \in \dot{\bigcup}_{i \in \{1,\ldots,m\}} J_i$ returns the index $i \in \{1, \ldots, m\}$ such that $y \in J_i$. Then the time complexity of the machine M is $\Omega(\log_2(m))$.*

We omit the proof since its main idea the same as the idea of the proof of Lemma 2.17. In particular, it is easy to observe that the input-output map of the machine M from the lemma must be piecewise constant. In addition, we know that it is piecewise rational. And a rational function which is constant on an open subinterval of $\mathbb{R}$ is a constant function by Lemma 1.18.

Recall that the connected components J_i are not always disjoint. In addition, some of them might be points and not open intervals. The following corollary states a lower bound for a general set of components.

Corollary 4.8. *Let $J_1, \dots, J_m$ be open intervals and points. Let $\ell \in \mathbb{N}$ be the minimal natural number such that there exist open intervals $J'_1, \dots, J'_\ell$ with the following properties. The intervals $J'_1, \dots, J'_\ell$ are disjoint, the inclusion*

$$\left(\dot{\bigcup_{j \in \{1,\dots,\ell\}}} J'_j \right) \subseteq \left(\bigcup_{i \in \{1,\dots,m\}} J_i \right)$$

holds and the set

$$\left(\bigcup_{i \in \{1,\dots,m\}} J_i \right) \setminus \left(\dot{\bigcup_{j \in \{1,\dots,\ell\}}} J'_j \right)$$

is a finite set.

Let M be a BSS machine such that given a value $y \in \bigcup_{i \in \{1,\dots,m\}} J_i$ returns an index $i \in \{1, \dots, m\}$ such that $y \in J_i$. Then the time complexity of the machine M is $\Omega(\log_2(\ell))$.

4.4 Step 2: Approximating a root of a rational function

In the previous section we showed how to find an interval $J_{\gamma,i}$ which contains a given value $y = \Phi_{M_0}(x)$ for some $x \in \Omega_{M_0}$. Therefore, we can now assume that an adversary knows the path γ, the corresponding rational function φ_γ, the component $I_{\gamma,i}$, its endpoints $a_{\gamma,i}, b_{\gamma,i}$ and the component $J_{\gamma,i}$ such that $y \in J_{\gamma,i}$. For shorter notation we will write φ, a, b, I, J instead of $\varphi_\gamma, a_{\gamma,i}, b_{\gamma,i}, I_{\gamma,i}, J_{\gamma,i}$ in this section.

First observe that if the component J is a point then we are done. We know that $\varphi(I) = J$ and therefore the component I is either a point or φ is a constant function. Thus, the machine M only outputs the value a (one of the endpoints of I), which is in the set of machine constants. Hereafter, we assume that J and I are open intervals.

Now an adversary has to find $x' \in I$ such that $|\varphi(x') - y| < \epsilon$. In other words, he has to construct a BSS machine which inverts the rational function φ on the open interval I with the precision ϵ.

There is a wide range of algorithms which approximate a zero of a polynomial (respectively, rational function). The common problem of these algorithms is how to choose a good starting point or starting points. It is usually not obvious that there exists an algorithm making this choice. We were facing this problem in the previous chapter in which we were inverting the polynomial $x^2 - y$ on the interval $[0, \infty)$. The situation in the general case is even more complicated.

The authors of the book [1] presented a general method how to find a good starting point for Newton's algorithm. The method uses some non-trivial results from homotopy theory. Since we do not need this theory later in our work, we present a different method of approximating a zero of a polynomial (respectively, rational function). Our method is based on the half-interval search algorithm which is not as efficient as Newton's method; however, it suffices for our purposes.

Let us emphasis, that we want to present a general method that can be applied on every rational function φ and every interval I such that φ is continuous on I. Moreover, we want to study the time complexity of the presented method. That is why this section consists of many auxiliary statements and technical computations.

For better orientation, let us first explain the idea of our method. We divide the problem of finding x' in two parts.

1. Find starting points $x_{min}, x_{max} \in I \cup \{a, b\}$ such that

$$\varphi(x_{min}) \leq y \leq \varphi(x_{max}). \tag{4.1}$$

   This part is discussed in Subsection 4.4.1.

2. Find x' from the closed interval with endpoints x_{min}, x_{max} such that

$$|\varphi(x') - y| < \epsilon,$$

   for the fixed $\epsilon > 0$. As we have already mentioned, we use the half-interval search algorithm to solve this problem. Details are studied in Subsection 4.4.2.

In the following remark we discuss why points x_{min}, x_{max} satisfying inequalities (4.1) are indeed good starting points for the half-interval search algorithm.

Remark 4.9. We know that the rational function φ is continuous on the interval I. From the Intermediate Value Theorem we know that there exists a point $x_Y \in I$ such that $\varphi(x_Y) = y$ and the value x_Y is contained in the interval with endpoints x_{min}, x_{max}. In particular, it holds that

$$\begin{aligned} x_{min} \leq x_Y \leq x_{max}, &\quad \text{if} \quad x_{min} < x_{max} \text{ and} \\ x_{max} \leq x_Y \leq x_{min}, &\quad \text{if} \quad x_{max} < x_{min}. \end{aligned}$$

The next remark discusses a special (but very important) case we have to be aware of later in this section.

Remark 4.10. If $x_{min} \in \{a, b\}$ then the value $\varphi(x_{min})$ might not be defined. In such a case $\varphi(x_{min})$ in the inequality (4.1) stands for either $\lim\limits_{x \to a^+} \varphi(x)$ or $\lim\limits_{x \to b^-} \varphi(x)$.

The same applies for x_{max}.

We do cover this special case in our algorithm although it requires some technical arguments which make the method less intuitive.

4.4.1 Finding good starting points

The rational function φ is continuous on the interval I. Thus, an adversary can precompute the points of global maximum and minimum of the rational function φ on the interval I (they are constants of the inverting machine M). Observe that if x_{min} is a point of global minimum and x_{max} is a point of global maximum then the inequalities (4.1) are satisfied. So did we solve the problem of finding good starting points?

The answer is: partially. The problem is that global extremes of a rational function φ on an interval I might not exist. Recall that the interval I is an open interval.

This is the reason why we will define three auxiliary values and prove several technical statements before we present an algorithm that finds the points x_{min}, x_{max}. For better orientation, we now discuss the main idea of the algorithm.

We first define auxiliary points $g_{min}, g_{max} \in I \cup \{a, b\}$. They will be either points of a global extreme of the function φ on the interval I or endpoints of the interval I. Secondly, we define a value $L > 0$ such that for every $x \in I$ the following implication holds

$$\varphi(x) = y \;\Rightarrow\; -L \leq x \leq L.$$

In other words, the value L bounds the absolute value of the zeros of the rational function $\varphi(x) - y$.

Using these auxiliary values, we define x_{min}, x_{max} in such a way that inequalities (4.1) hold. In addition, we prove a bound on the distance of the starting points. Concretely, we show that

$$|x_{min} - x_{max}| \leq 2L.$$

This bound will be important for the time complexity analysis of the half-interval search algorithm.

Auxiliary results

Let us start by defining the auxiliary points g_{min}, g_{max}.

Definition 4.11. Let I be an interval with endpoints a, b. For a rational function φ that is continuous on I we define the value g_{min} as follows.

If the global minimum of φ on I exists then set g_{min} to the point of minimal value. Otherwise, at least one of the following cases is true:

$$\forall c \in I\colon \lim_{x \to a^+} \varphi(x) < \varphi(c),$$

then we define $g_{min} = a$, or

$$\forall c \in I\colon \lim_{x \to b^-} \varphi(x) < \varphi(c)$$

and in this case we set $g_{min} = b$. If both statements are true then we set $g_{min} = \min\{|a|, |b|\}$.

Analogously, we define the value g_{max}. It is either a point of maximal value or one of the endpoints of the interval I.

Let us now make few remarks about these values.

Remark 4.12.

1. The values g_{min}, g_{max} do not depend on the value y that is given to the adversary. Thus, the computation of these values can be done in the pre-computation phase (they are constants of the inverting machine M).
2. It is clear from the definition that $g_{min}, g_{max} \in I \cup \{a, b\}$.
3. If g is a point of global extreme of the rational function φ on the interval I then it is also a point of global extreme of the rational function

$$\varphi(x) - y = \frac{p(x) - y \cdot q(x)}{q(x)}$$

on interval the I, where $p, q \in \mathbb{R}[x]$ is the fixed pair of polynomials whose quotient represents the rational function φ (recall Remark 2.2).

As a next step we establish the notation for polynomials p, q and r, where the polynomial r is defined as $r = p - yq$. Note that y is treated as a parameter here, not as a variable. We denote

$$\begin{aligned} p(x) &= \sum_{i=0}^{d_p} p_i x^i, \\ q(x) &= \sum_{i=0}^{d_q} q_i x^i, \\ r(x) &= p(x) - yq(x) = \sum_{i=0}^{d} r_i x^i, \end{aligned}$$

where $p_i, q_i, r_i \in \mathbb{R}$ and $p_{d_p}, q_{d_q}, r_d \neq 0$. Recall from Section 1.2 the following notation

$$\begin{aligned} \max_p &= \max\{|p_i| \colon i \in \{0, \dots d_p\}\}, \\ \max_q &= \max\{|q_i| \colon i \in \{0, \dots d_q\}\}, \\ \max_r &= \max\{|r_i| \colon i \in \{0, \dots d\}\}. \end{aligned}$$

Since $\max_{yq} = y \cdot \max_q$, we know by Observation 3.16 that

$$\max_r \leq \max_p + y \cdot \max_q.$$

Let us define the value $L \in \mathbb{R}^+$ as

$$L := 1 + \frac{1}{|r_d|} \cdot (\max_p + y \cdot \max_q). \tag{4.2}$$

Lemma 4.13. *For the value L defined above it holds*

$$\left\{ x \in I \colon \frac{p(x) - y \cdot q(x)}{q(x)} = 0 \right\} \subseteq [-L, L].$$

Proof. From Lemma 1.16 we know that for every $\alpha \in I$ such that $r(\alpha) = 0$

$$|\alpha| \leq 1 + \frac{1}{|r_d|} \cdot \max\{|r_i| \colon i \in \{0, \dots d-1\}\}.$$

Note that

$$\max\{|r_i| \colon i \in \{0, \dots d-1\}\} \leq \max_r \leq \max_p + y \cdot \max_q.$$

This implies that for every $\alpha \in I$ such that $r(\alpha) = 0$ we have

$$|\alpha| \leq 1 + \frac{1}{|r_d|} \cdot (\max_p + y \cdot \max_q).$$

Since the expression on the right hand side equals L, we can equivalently write

$$\{x \in I \colon p(x) - y \cdot q(x) = 0\} \subseteq [-L, L].$$

The rational function φ is defined on the interval I; thus, the value $q(x)$ is non-zero for every $x \in I$. This implies that

$$\left\{x \in I \colon \frac{p(x) - y \cdot q(x)}{q(x)} = 0\right\} = \{x \in I \colon p(x) - y \cdot q(x) = 0\}.$$

Therefore, we can conclude that

$$\left\{x \in I \colon \frac{p(x) - y \cdot q(x)}{q(x)} = 0\right\} \subseteq [-L, L].$$

□

One of the consequences of the lemma is that

$$\left\{x \in I \colon \frac{p(x) - y \cdot q(x)}{q(x)} = 0\right\} \subseteq (I \cup \{a, b\}) \cap [-L, L].$$

In addition, we know that $y \in J = \varphi(I)$. Thus, there exists at least one $\alpha \in I$ such that $\varphi(\alpha) - y = 0$. This implies that

$$\left\{x \in I \colon \frac{p(x) - y \cdot q(x)}{q(x)} = 0\right\}$$

is a non-empty set and therefore we can make the following remark.

Remark 4.14. The intersection $(I \cup \{a, b\}) \cap [-L, L]$ is not empty.

The next lemma motivates the definition of the points x_{min}, x_{max}.

Lemma 4.15. *Let L, g_{min}, g_{max} be values defined as above.*

1. *If $g_{min} \notin [-L, L]$ then*

$$\varphi(-L) \leq y \quad or \quad \varphi(L) \leq y.$$

Moreover, the following two implications are true

$$\varphi(-L) > y \Rightarrow a \leq L < g_{min} \leq b,$$
$$\varphi(L) > y \Rightarrow a \leq g_{min} < -L \leq b.$$

2. *Analogously, if $g_{max} \notin [-L, L]$ then at least one of the following inequalities holds*
$$\varphi(-L) \geq y \quad or \quad \varphi(L) \geq y.$$
Moreover, the following two implications are true
$$\varphi(-L) < y \Rightarrow a \leq L < g_{max} \leq b,$$
$$\varphi(L) < y \Rightarrow a \leq g_{max} < -L \leq b.$$

Proof. We prove the lemma only for g_{min} since the proof for g_{max} is analogous.

Let $g_{min} \notin [-L, L]$ and suppose, for the sake of contradiction, that both
$$\varphi(-L) - y > 0 \quad \text{and} \quad \varphi(L) - y > 0. \tag{4.3}$$
Recall from Corollary 1.17 that the value L is such that either
$$\forall x > L\colon\ \varphi(x) - y > 0 \quad \text{or} \quad \forall x > L\colon\ \varphi(x) - y < 0.$$
Similarly, either
$$\forall x < -L\colon\ \varphi(x) - y > 0 \quad \text{or} \quad \forall x < -L\colon\ \varphi(x) - y < 0.$$
Observe that the following implication must hold
$$\varphi(L) - y > 0 \Rightarrow \forall x > L\colon\ \varphi(x) - y > 0 \tag{4.4}$$
since otherwise we would yield a contradiction with the definition of L by the Intermediate Value Theorem. We can argue similarly in the case other case; i.e. when $\varphi(-L) - y > 0$. Thus, from assumptions (4.3) we can derive that we are in the situation
$$\forall x > L\colon\ \varphi(x) - y > 0 \quad \text{and} \quad \forall x < -L\colon\ \varphi(x) - y > 0.$$
Since $g_{min} \notin [-L, L]$, we have
$$\varphi(g_{min}) - y > 0.$$
But that contradicts the definition of g_{min} since we know that there exists at least one $\alpha \in [-L, L]$ such that $\varphi(\alpha) - y = 0$.

Let us focus on the second part of the statement. From Remark 4.12 we know that $a \leq g_{min} \leq b$ and we observed in Remark 4.14 that the intersection $(I \cup \{a, b\}) \cap [-L, L]$ is non-empty. Therefore, if $g_{min} \notin [-L, L]$ then either
$$g_{min} < -L \Rightarrow a \leq g_{min} < -L \leq b$$
or we have
$$L < g_{min} \Rightarrow a \leq L < g_{min} \leq b.$$
If $\varphi(-L) > y$ then by implication (4.4) we have
$$\forall x < -L\colon\ \varphi(x) > y.$$
It follows that $L < g_{min}$ must be the case because otherwise we would achieve a contradiction with the definition of g_{min} as in the first part of the proof. Let us conclude our reasoning:
$$\varphi(-L) > y \overset{(4.4)}{\Longrightarrow} L < g_{min} \overset{\text{Def. 4.11}}{\Longrightarrow} a \leq L < g_{min} \leq b.$$
For analogous reasons, if $\varphi(L) > y$ then $g_{min} < -L$. □

Starting points

We are now finally prepared to define the starting points x_{min}, x_{max}.

Definition 4.16. Let the values g_{min}, g_{max} and L be as defined above. Then we define the points x_{min}, x_{max} as

$$x_{min} = \begin{cases} g_{min}, & \text{if } g_{min} \in [-L, L], \\ -L, & \text{if } g_{min} \notin [-L, L],\ a \leq g_{min} < -L \leq b \text{ and } \varphi(-L) \leq y, \\ L, & \text{otherwise;} \end{cases}$$

$$x_{max} = \begin{cases} g_{max}, & \text{if } g_{max} \in [-L, L], \\ -L, & \text{if } g_{max} \notin [-L, L],\ a \leq g_{max} < -L \leq b \text{ and } \varphi(-L) \geq y, \\ L, & \text{otherwise.} \end{cases}$$

Recall from the beginning of this subsection that we wanted to find values x_{min}, x_{max} such that

$$\varphi(x_{min}) \leq y \leq \varphi(x_{max}).$$

As discussed in Remark 4.10, $\varphi(x_{min})$ in the inequality above stands for either $\lim\limits_{x \to a^+} \varphi(x)$ or $\lim\limits_{x \to b^-} \varphi(x)$ if the value $\varphi(x_{min})$ is not defined. The same holds for the point x_{max}.

In the next lemma we prove that we defined the starting points x_{min}, x_{max} in Definition 4.16 correctly.

Lemma 4.17. *The values x_{min}, x_{max} from Definition 4.16 satisfy the inequalities (4.1). Moreover, it holds that*

$$x_{min}, x_{max} \in (I \cup \{a, b\}) \cap [-L, L].$$

As a consequence, we have the following bound

$$|x_{min} - x_{max}| \leq 2L.$$

Proof. We will prove the lemma only for x_{min} since the arguments are analogous in the case of x_{max}.

We prove the statements by case analysis splitting on the possible definition of the starting point x_{min}.

1. If $g_{min} \in [-L, L]$ then $x_{min} = g_{min}$. By definition of g_{min} we have

   $$\forall c \in I: \ \varphi(x_{min}) < \varphi(c).$$

   Since $y \in J = \varphi(I)$, the desired inequality follows. Recall from Remark 4.10 that $\varphi(x_{min})$ in the inequality above stands for either $\lim\limits_{x \to a^+} \varphi(x)$ or $\lim\limits_{x \to b^-} \varphi(x)$ if the value $\varphi(x_{min})$ is not defined.

   From Remark 4.12 we know that $g_{min} \in I \cup \{a, b\}$. Therefore, it is clear that $x_{min} \in (I \cup \{a, b\}) \cap [-L, L]$ in this case.

2. If $g_{min} \notin [-L, L], \varphi(-L) \le y$ and $a \le g_{min} < -L \le b$ then the value x_{min} is defined as $x_{min} = -L$. Thus, we see directly that

$$\varphi(x_{min}) = \varphi(-L) \le y \quad \text{and} \quad x_{min} = -L \in (I \cup \{a, b\}) \cap [-L, L].$$

3. There are two situations in which the value x_{min} is defined as L. In both of them $g_{min} \notin [-L, L]$.

   - If $\varphi(-L) > y$ then we know from Lemma 4.15 that $\varphi(L) \le y$ must be the case. In addition, from the same lemma we know that

     $$\varphi(-L) > y \Rightarrow a \le L < g_{min} \le b.$$

     Thus, $x_{min} = L \in (I \cup \{a, b\}) \cap [-L, L]$.
   - In the second case we have $\varphi(-L) \le y$ but $a \le g_{min} < -L \le b$ is not the case. Then from Lemma 4.15 we know that

     $$\varphi(L) \le y.$$

     In addition, the fact that $a \le g_{min} < -L \le b$ does not hold implies that $L < g_{min}$. The intersection $(I \cup \{a, b\}) \cap [-L, L]$ is non-empty by Remark 4.14. Moreover, by Remark 4.12 we know that $g_{min} \in I \cup \{a, b\}$. As a result,

     $$a \le L < g_{min} \le b.$$

     Hence, $x_{min} = L \in (I \cup \{a, b\}) \cap [-L, L]$ in this case as well.

□

Definition 4.16 gives us an algorithm how to find the values x_{min}, x_{max} knowing the values g_{min}, g_{max} and L. The next lemma states the overall time complexity of finding the starting points x_{min} and x_{max}.

Lemma 4.18. *The time complexity of finding x_{min}, x_{max} in the way described above is $\mathcal{O}(T)$, where T is the time bound of the machine M_0 which computes the function φ on the interval I.*

Proof. We observed in Remark 4.12 that the computation of g_{min}, g_{max} can be done in the preparation phase. In addition, we assume that an adversary knows the polynomials p and q defining the rational function φ. That means that the leading coefficients $\mathrm{lc}(p)$, $\mathrm{lc}(q)$, the degrees $\deg(p)$, $\deg(q)$ and the values $\max_p$, $\max_q$ are constants of the inverting machine M.

There are at most 2 arithmetic operations needed in order to compute the coefficient r_d since

$$r_d = \begin{cases} \mathrm{lc}(p), & \text{if } d_p > d_q, \\ \mathrm{lc}(q), & \text{if } d_p < d_q, \\ \mathrm{lc}(p) - y \cdot \mathrm{lc}(q), & \text{if } d_p = d_q. \end{cases}$$

The value L is defined as

$$L = 1 + \frac{1}{|r_d|} \cdot (\max_p + y \cdot \max_q).$$

We see that the adversary can compute the auxiliary value L in constant time.

Finally, only constantly many branch nodes and one computation of $\varphi(-L)$ are needed in order to decide how to define the starting points x_{min} and x_{max} following the algorithm from Definition 4.16.

Putting all together, we can conclude that the time complexity is $\mathcal{O}(T)$ as stated in the lemma. □

Before we continue to the final part of this section, i.e. approximating the value $x \in I$ given the value $y = \varphi(x)$, let us recall the special case from Remark 4.10 and define two additional values f_{min}, f_{max} as follows

$$f_{min} = \begin{cases} \varphi(x_{min}), & \text{if } \varphi(x_{min}) \text{ is defined,} \\ -\infty, & \text{otherwise;} \end{cases}$$

$$f_{max} = \begin{cases} \varphi(x_{max}), & \text{if } \varphi(x_{max}) \text{ is defined,} \\ \infty, & \text{otherwise.} \end{cases}$$

The rational function φ is continuous and defined on the interval I. Therefore, the following implication holds

$$x_{min} \in I \Rightarrow \varphi(x_{min}) > -\infty.$$

Since $x_{min} \in I \cup \{a, b\}$, the value $\varphi(x_{min})$ is undefined if only if one of the following cases is true

$$\begin{aligned} x_{min} = a \quad &\text{and} \quad \lim_{x \to a^+} \varphi(x) = -\infty, \\ x_{min} = b \quad &\text{and} \quad \lim_{x \to b^-} \varphi(x) = -\infty. \end{aligned}$$

We can make an analogous discussion about $\varphi(x_{max})$.

Lemma 4.19. *The time complexity of computing the values f_{min}, f_{max} is $\mathcal{O}(T)$ where T is the time bound of the machine M_0 computing the rational function φ on the interval I.*

Proof. To compute the value f_{min} we create a new machine M_0' such that computes the rational function φ on inputs from interval I as the machine M_0, i.e. it preserves the halting path γ. We know that every branch node on the path γ has two successor nodes. One of them corresponds to the next node of the path γ. We define the other one to be the output node which sets the output of the machine to $-\infty$. We know that the number of elementary instructions and branching nodes on the path γ is less or equal to T. From our construction it is clear that the path γ is the longest path of machine M_0'. Thus, the time complexity of the machine M_0' is T.

We can proceed analogously in the case of f_{max}. □

In fact, the lemma above is not formally correct. The problem is that no BSS machine can store (or output) the symbols "$-\infty$", "∞". However, this problem can be solve by the following programming technique. We represent f_{min} as a pair

$$(\text{defined}, \text{value}) = \begin{cases} (1, \varphi(x_{min})), & \text{if } \varphi(x_{min}) \text{ is defined,} \\ (0, 0), & \text{otherwise .} \end{cases}$$

Thus, if we ask whether the equality “$f_{min} = -\infty$” holds then we ask in fact if “defined $= 0$” is true.

Let us conclude this subsection with a corollary that follows from Lemma 4.17 and Lemma 4.19.

Corollary 4.20. *The auxiliary values $x_{min}, x_{max}, f_{min}, f_{max}$ can be computed by a constant time BSS machine.*

4.4.2 The half-interval search algorithm

Let us first recapitulate the current situation. An adversary is given a value $y = \varphi(x)$, where φ is a rational function which is continuous on the open interval I. His aim is to find a value $x' \in I$ such that

$$|\varphi(x') - y| < \epsilon,$$

where $\epsilon > 0$ is the fixed precision. Equivalently, we want to construct a BSS machine that inverts the rational function φ on the interval I.

Algorithm 4.2 defines such a BSS machine M. As we have already mentioned earlier in this chapter, the algorithm is based on the half-interval search method.

```
Input: $y \in J$
Output: $x' \in I$ such that $|\varphi(x') - y| < \epsilon$
$x_L \leftarrow x_{min}$;
$f_L \leftarrow f_{min}$;
$x_U \leftarrow x_{max}$;
$f_U \leftarrow f_{max}$;
while $f_L = -\infty$ or $f_U = \infty$ or $|f_U - f_L| \geq \epsilon$ do
    $x_M = \frac{1}{2}(x_U + x_L)$;
    $f_M = \varphi(x_M)$;
    if $f_M \geq y$ then
        $x_U \leftarrow x_M$;
        $f_U \leftarrow f_M$;
    end
    else
        $x_L \leftarrow x_M$;
        $f_L \leftarrow f_M$;
    end
end
return $x_U$
```

Algorithm 4.2: Inversion of a rational function φ on an interval I with the precision $\epsilon > 0$

Note, that Algorithm 4.2 is defined for a concrete function φ and concrete values $x_{min}, x_{max}, f_{min}, f_{max}$. They are assumed to be part of the machine constants. This is a reasonable approach since all of these values are known at this stage of the process. In Section 4.3 we constructed a constant time BSS machine which determines the rational function φ and the interval I. In Subsection 4.4.1 proved that there is a constant time BSS machine which finds the starting points x_{min} and x_{max} and Lemma 4.19 tells us that there is a constant time machine which defines the values f_{min} and f_{max}.

Thus, the Algorithm 4.2 can be understood as a subroutine of a bigger algorithm. Namely, an algorithm which

1. executes Algorithm 4.1,
2. finds the values x_{min}, x_{max} as described in Definition 4.16,
3. defines the values f_{min}, f_{max} by running the algorithm from the proof of Lemma 4.19 and finally
4. executes the corresponding variant of Algorithm 4.2.

Let us now focus on the correctness and the time complexity of Algorithm 4.2

Correctness of Algorithm 4.2

Let us first state two auxiliary lemmas. that will be useful in the proof of the correctness of Algorithm 4.2.

Lemma 4.21. *In every step of Algorithm 4.2 the following statements are true.*

1. *The following inequalities hold*
$$f_L \leq y \leq f_U. \tag{4.5}$$
2. *The value $f_U < \infty$ if and only if $\varphi(x_U)$ is defined and $f_U = \varphi(x_U)$.*
3. *The value $f_L > -\infty$ if and only if $\varphi(x_L)$ is defined and $f_L = \varphi(x_L)$.*

Proof. 1. From the definition of values x_{min}, x_{max} we know that inequalities (4.5) holds after the initialization step. Suppose the inequalities (4.5) holds before a while-loop and let us denote f_L', f_U' the values f_L, f_U after the while-loop. Let us discuss both possible cases.

- If $f_M \geq y$ then $f_L' = f_L$, thus $f_L' = f_L \leq y$. The value f_U' is set to f_M which is greater or equal to y. Hence, $f_L' \leq y \leq f_U'$.
- If $f_M < y$ then $f_U' = f_U \geq y$ and $f_L' = f_M$. So in this case we achieve the desired inequalities $f_L' \leq y \leq f_U'$ as well.

2. In the while-loop in which the values x_U, f_U are reassigned it holds
$$f_U = f_M = \varphi(x_M) = \varphi(x_U) < \infty.$$
Thus, we need to study only the case when $f_U = f_{max}$. If $f_{max} < \infty$ then by definition of f_{min} the value $\varphi(x_{max})$ must be defined and $f_{max} = \varphi(x_{max})$. The other implication follows from the fact that if $\varphi(x_{max})$ is defined then $\varphi(x_{max}) < \infty$.

3. Analogous to the previous case.

□

Lemma 4.22. *Let $n \in \mathbb{N}$. Suppose that we perform n while-loops and $f_M < y$ is always the case (i.e. we are reassigning only the value x_L). Then*

$$x_U = x_{max} \quad \text{and} \quad x_L = \frac{x_{min} + x_{max} \cdot (2^n - 1)}{2^n}.$$

Proof. It is clear that $x_U = x_{max}$. We prove the stated form of x_L by induction on n. The statement is true for $n = 1$ since

$$x_L = \frac{x_{min} + x_{max}}{2}.$$

Let x_L be the value after $n-1$ while-loops and x'_L the value after n while-loop. Suppose that

$$x_L = \frac{x_{min} + x_{max} \cdot (2^{n-1} - 1)}{2^{n-1}}.$$

Then using the induction hypothesis we can derive

$$x'_L = \frac{x_{max} + x_L}{2} \overset{IH}{=} \frac{x_{min} + 2 \cdot 2^{n-1} \cdot x_{max} - x_{max}}{2 \cdot 2^{n-1}} = \frac{x_{min} + x_{max} \cdot (2^n - 1)}{2^n}.$$

□

Now we are prepared to prove that Algorithm 4.2 works correctly.

Theorem 4.23. *Algorithm 4.2 halts on every input $y \in J$ and its output x' satisfies*

$$|\varphi(x') - y| < \epsilon.$$

Proof. We start with the correctness of the output. The algorithm halts when

$$f_L > -\infty, \ \ f_U < \infty, \ \text{ and } |f_U - f_L| < \epsilon.$$

We know from Lemma 4.21 that the inequalities

$$f_L \leq y \leq f_U$$

hold in every step of the algorithm; specially, in the last step. Moreover, from the Lemma 4.21 we also know that

$$f_U = \varphi(x_U)$$

must be the case since $f_U < \infty$. Thus, we can conclude

$$|\varphi(x_U) - y| = |f_U - y| \leq |f_U - f_L| < \epsilon.$$

Now we want to prove that the algorithm always halts. Let us first show that after finitely many steps we are in the situation that

$$x_L, x_U \in I \quad \text{and} \quad f_L = \varphi(x_L) > -\infty \quad \text{and} \quad f_U = \varphi(x_U) < \infty. \tag{4.6}$$

In every while-loop exactly one of the values x_L, x_U is reassigned. Hence, without lost of generality, we can assume that x_L is reassigned in the first while-loop. Consequently, after the first while-loop

$$f_L = f_M = \varphi(x_M) = \varphi(x_L). \tag{4.7}$$

Because $x_M \in I$, we know that $f_M > -\infty$.

The value f_U does not change in the first while-loop, i.e. $f_U = f_{max}$. According to Lemma 4.21 either

$$f_U = f_{max} = \varphi(x_{max}) = \varphi(x_U) < \infty,$$

and we proved (4.6), or

$$f_U = f_{max} = \infty \quad \text{and } x_{max} \in \{a, b\}.$$

Suppose that we are in the situation in which $x_{max} = b$. This means that

$$\lim_{x \to b^-} \varphi(x) = \infty.$$

Recall that $x_{max} \in [-L, L]$; hence, $x_{max} \in \mathbb{R}$. Therefore, by definition of a limit we can equivalently write

$$\forall K > 0 \; \exists \delta > 0 \; \forall x \in I \colon |b - x| < \delta \; \Rightarrow \; \varphi(x) > K.$$

So for $K = |y|$ there exists $\delta_1 > 0$ such that

$$\forall x \in I \colon \; |b - x| < \delta_1 \; \Rightarrow \; \varphi(x) > |y|. \tag{4.8}$$

Our aim is to show that there exists $n \in \mathbb{N}$ such that after n while-loops the value x_U must have been reassigned at least once. The idea is to show that if the value x_U never changes then we achieve a contradiction with the inequality $\varphi(x_L) \le y$ from Lemma 4.21.

From Lemma 4.22 we know that after n while-loops, where only x_L has being reassigned, the values x_U, x_L are

$$x_U = b \quad \text{and} \quad x_L = \frac{x_{min} + b \cdot (2^n - 1)}{2^n}.$$

Let us express the difference between x_L and x_U as

$$|x_U - x_L| = \left| b - \frac{x_{min} + b \cdot (2^n - 1)}{2^n} \right| = \left| \frac{b - x_{min}}{2^n} \right|.$$

From Lemma 4.17 we know that $a \le x_{min} \le b$, therefore

$$|x_U - x_L| = \frac{b - x_{min}}{2^n}.$$

We can derive the following equivalence

$$|x_U - x_L| < \delta_1 \; \Leftrightarrow \; \log_2(b - x_{min}) - \log_2(\delta_1) < n.$$

Thus, after $\lfloor \log_2(b - x_{min}) - \log_2(\delta_1) \rfloor + 1$ while-loops, in which only the value x_L has being reassigned, we have according to (4.8) the following implication

$$|b - x_L| < \delta_1 \ \Rightarrow \varphi(x_L) > |y| \geq y. \tag{4.9}$$

From equation (4.7) we know that $f_L = \varphi(x_L)$. But then the implication (4.9) contradicts the first statement of Lemma 4.21, i.e. that in every step of the algorithm

$$f_L \leq y \leq f_U.$$

Thus, we proved that the values x_U, f_U must be eventually reassigned. Consequently,

$$f_U = f_M = \varphi(x_M) = \varphi(x_U),$$

where $x_M \in I$ and $f_M < \infty$. We omit the proof of (4.6) for $x_{max} = a$ since the idea is the same. We only conclude that in both cases we are in the situation expressed by (4.6) after at most

$$\lfloor \log_2(L) - \log_2(\delta_1) \rfloor + 2$$

while-loops. Recall from Lemma 4.17 that $|x_{min} - x_{max}| \leq 2L$.

Now we are in the situation that (4.6) holds. We know that the rational function φ is continuous on the interval I; thus, by the definition of continuity there exists $\delta > 0$ such that for $x_L, x_U \in I$

$$|x_U - x_L| < \delta \Rightarrow |\varphi(x_U) - \varphi(x_L)| < \epsilon.$$

This implies that there exists $\delta_2 > 0$ such that

$$|x_U - x_L| < \delta_2 \Rightarrow (|f_U - f_L| < \epsilon \wedge f_U < \infty \wedge f_L > -\infty) \Rightarrow \text{ algorithm halts.}$$

We know from Lemma 4.17 that the initial difference is

$$|x_{max} - x_{min}| \leq 2L.$$

In every while-loop the difference decreases by factor $\frac{1}{2}$. Thus, after $m \in \mathbb{N}$ while-loops the difference is

$$\frac{|x_{max} - x_{min}|}{2^m} \leq \frac{2L}{2^m}.$$

The next equivalence expresses how many while-loops we need to achieve a difference smaller than δ_2. In particular,

$$\frac{2L}{2^m} < \delta_2 \ \Leftrightarrow \ 1 + \log_2(L) - \log_2(\delta_2) < m.$$

We derived that after at most $\lfloor \log_2(L) - \log_2(\delta_2) \rfloor + 2$ while-loops the algorithm halts assuming that the conditions (4.6) are fulfilled.

Let us summarize our proof. Firstly, we proved that there exists $\delta_1 > 0$ such that after at most

$$\lfloor \log_2(L) - \log_2(\delta_1) \rfloor + 2$$

while-loops the state of the algorithm satisfies the conditions (4.6). Secondly, we showed that there exists $\delta_2 > 0$ such that the algorithm halts after at most

$$\lfloor \log_2(L) - \log_2(\delta_2) \rfloor + 2$$

additional while-loops. $\square$

Table 4.1: Steps of Algorithm 4.2 for the rational function $\varphi(x) = \frac{x^2}{1-x}$, precision $\epsilon = 1$ and input $y = 20$

	x_L	x_U	f_L	f_U	$\lvert f_U - f_L\rvert$
0	0	1	0	∞	✗
1	0.5	1	0.5	∞	✗
2	0.75	1	2.25	∞	✗
3	0.875	1	6.125	∞	✗
4	0.9375	1	14.0625	∞	✗
5	0.9375	0.96875	14.0625	30.0313	15.9688
6	0.953125	0.96875	19.3802	30.0313	10.651
7	0.953125	0.960938	19.3802	23.6391	4.25885
8	0.953125	0.957031	19.3802	21.3157	1.93549
9	0.953125	0.955078	19.3802	20.3058	0.925583

A concrete example

To illustrate the theory introduced in this section, let us apply it on one concrete example.

Example 4.24. Let us consider the rational function $\varphi(x) = \frac{p(x)}{q(x)} = \frac{x^2}{1-x}$ on the interval $I = (-\infty, 1)$. Suppose that the input value is $y = 20$ and the fixed precision is $\epsilon = 1$.

We see that the point of global minimum of φ on I is 0 and the point of global maximum does not exists. Thus, according to Definition 4.11 we have

$$g_{min} = 0, \;\; g_{max} = 1.$$

We can compute the value L defined in (4.2) as

$$\begin{aligned} L &= 1 + \frac{1}{|\mathrm{lc}(p + yq)|} \cdot (\max_p + y \cdot \max_q) \\ &= 1 + \frac{1}{|1|} \cdot (1 + 20 \cdot 1) \\ &= 22 \end{aligned}$$

We see that both $g_{min} \in [-L, L]$ and $g_{max} \in [-L, L]$. Thus, by Definition 4.16 the values x_{min}, x_{max} are defined as

$$x_{min} = g_{min} = 0, \;\; x_{max} = g_{max} = 1.$$

It remains to assign the values f_{min}, f_{max}. They are in this case

$$f_{min} = \frac{x_{min}^2}{1 - x_{min}} = 0, \;\; f_{max} = \infty.$$

Table 4.1 describes the computation of Algorithm 4.2.

Note that both

$$\lim_{x\to-\infty} \varphi(x) = \infty \quad \text{and} \quad \lim_{x\to 1^-} \varphi(x) = \infty.$$

Table 4.2: Steps of Algorithm 4.2 for the rational function $\varphi(x) = \frac{x^2}{1-x}$, fixed precision $\epsilon = 1$, input $y = 20$ and an alternative choice of g_{max}

	x_L	x_U	f_L	f_U	$\lvert f_U - f_L \rvert$
0	0	-22	0	21.0435	21.0435
1	-11	-22	10.0833	21.0435	10.9601
2	-16.5	-22	15.5571	21.0435	5.48634
3	-19.25	-22	18.2994	21.0435	2.7441
4	-20.625	-22	19.6712	21.0435	1.37224
5	-20.625	-21.3125	19.6712	20.3573	0.686075

The convention settled in Definition 4.11 gives us the choice $g_{max} = 1$. Nevertheless, the choice $g_{max} = -\infty$ would be also possible. Let us present the computation of the algorithm with this alternative choice of g_{max}.

The values x_{min} and f_{min} does not change whereas the values x_{max} and f_{max} are defined differently in this case; concretely, as

$$x_{max} = -L = -22, \;\; f_{max} = \varphi(x_{max}) = \frac{484}{23}.$$

Table 4.2 describes the computation of Algorithm 4.2 with this alternative choice of g_{max}.

We see from Table 4.2 that for this particular example it would be more efficient to choose $g_{max} = -\infty$ rather than $g_{max} = 1$.

Complexity analysis of Algorithm 4.2

We end this subsection with a discussion about the complexity of Algorithm 4.2. In the proof of Theorem 4.23 not only did we prove that Algorithm 4.2 halts on every input but we also estimated the halting time of the corresponding machine. Concretely, we showed that for every rational function φ and interval I, such that the function φ is continuous on I, there exist values $L > 0$ and $\delta_1, \delta_2 > 0$ such that after at most

$$\lfloor \log_2(L) - \log_2(\delta_1) \rfloor + \lfloor \log_2(L) - \log_2(\delta_2) \rfloor + 4$$

while-loops the algorithm halts. Observe that there are finitely many branch nodes, one computation of φ and finitely many elementary instruction in every while-loop. Thus, the halting time is proportional to the number of while-loops multiplied by $\mathcal{O}(T)$. Recall that T is the bound on the number of elementary instructions and branch nodes needed to in order to compute the function φ on the interval I.

Let us now study the values L, δ_1 and δ_2. Our aim is to discuss how to find these values and show that the values L and δ_1 depends on the input value y whereas the value δ_2 does not.

As a result, we achieve the following observation.

Observation 4.25. *The BSS machine defined by Algorithm 4.2 is a constant time BSS machine if the rational function φ and the interval I are such that the interval $J = \varphi(I)$ is a bounded interval.*

Let us first focus on the value L. Recall, that it is defined as

$$L = 1 + \frac{1}{|r_d|} \cdot (\max_p + y \cdot \max_q),$$

where

$$r_d = \begin{cases} \mathrm{lc}(p), & \text{if } d_p > d_q, \\ \mathrm{lc}(q), & \text{if } d_p < d_q, \\ \mathrm{lc}(p) - y \cdot \mathrm{lc}(q), & \text{if } d_p = d_q. \end{cases}$$

See equation (4.2) for more details. For a concrete rational function φ, we can understand the value L as a function of y. Let us demonstrate this on an example.

Example 4.26. Assume the same function and interval as in Example 4.24, i.e. $\varphi(x) = \frac{p(x)}{q(x)} = \frac{x^2}{1-x}$ on the interval $I = (-\infty, 1)$. Then

$$r_d = 1, \;\; \max_p = 1, \;\; \max_q = 1.$$

Hence, the function L is in this case defined as

$$L(y) = 1 + (1 + y) = 2 + y.$$

In contrast to the value L, we did not present a way how to find the values δ_1, δ_2 for a rational function φ continuous on the interval I. In the proof of Theorem 4.23 we showed that these values exist; however, the proof was not constructive. Therefore, we will now study the constructive approach.

Let us recall the definition of the value δ_1. We are in the situation in which either $x_{min} \in \{a, b\}$ and $f_{min} = -\infty$ or $x_{max} \in \{a, b\}$ and $f_{max} = \infty$. We discuss only the situation when

$$x_{max} = b > 0 \;\text{ and }\; f_{max} = \lim_{x \to b^-} \varphi(x) = \infty.$$

The other cases are analogous.

From Remark 4.14 we know that $x_{max} \in [-L, L]$, where $L \in \mathbb{R}$. Thus, the starting point x_{max} is defined as $x_{max} = b < \infty$. Recall that $\delta_1 > 0$ is such that

$$\forall x \in I \colon |b - x| < \delta_1 \;\Rightarrow\; \varphi(x) > |y|.$$

In other words, we want to find $\delta_1 > 0$ such that for every $0 < \delta < \delta_1$ we have

$$\varphi(b - \delta) > |y|.$$

Intuitively, we can express the value δ_1 in terms of the input y and the endpoint b.

We omit the general analysis of this problem. However, we show how to find δ_1 in one concrete example. In fact, we use the same function and interval as in Example 4.24.

Example 4.27. Let $\varphi(x) = \frac{x^2}{1-x}$ and $b = 1$. We want to find δ_1 such that for every $0 < \delta < \delta_1 < b = 1$ it holds

$$\varphi(1 - \delta) > |y|.$$

If we evaluate the rational function φ, we achieve

$$\frac{1-2\delta+\delta^2}{1-(1-\delta)} > |y|.$$

This can be equivalently expressed as

$$1-\delta(2+|y|)+\delta^2 > 0.$$

Since $0<\delta<1$, we know that the inequality holds if

$$0<\delta<\frac{2+|y|-\sqrt{4|y|+y^2}}{2}.$$

Thus, we showed that in this concrete case

$$\delta_1(y) := \frac{2+|y|-\sqrt{4|y|+y^2}}{2}.$$

If the input is $y=20$, as in Example 4.24, then

$$\delta_1 = \frac{2+20-\sqrt{4\cdot 20+400}}{2} = 11-\sqrt{120} \doteq 0.0455488.$$

Finally, let us express

$$\lfloor \log_2(b-x_{min}) - \log_2(\delta_1)\rfloor + 1 = \lfloor -\log_2(\delta_1)\rfloor + 1 = 4+1 = 5$$

Thus, according to the proof of Theorem 4.23 (concretely the equation (4.9)), the value x_U must be changed latest in the fifth while-loop. This is, indeed, the case in our example, see Table 4.1.

We believe that this concrete example gave an idea how to find a function $\delta_1(y)$ for a rational function φ and endpoint b.

Let us now discuss the value $\delta_2>0$. Recall that at this point we assume starting points $x_L, x_U \in I$ such that

$$-\infty < \varphi(x_L) \le y \le \varphi(x_U) < \infty.$$

We know that the function φ is continuous on the interval I. Moreover, it is uniformly continuous on the closed interval with endpoints x_L, x_U. Let us denote this interval I'. By definition there exists $\delta_2>0$ such that

$$\forall x_1, x_2 \in I'\colon |x_1-x_2|<\delta_2 \;\Rightarrow\; |\varphi(x_1)-\varphi(x_2)|<\epsilon.$$

Note that the value δ_2 can be understood as a function of ϵ. The question is how to find such a function $\delta_2(\epsilon)$. As in the case of δ_1, we omit the general discussion; however, we present one concrete example on which we demonstrate the strategy of finding $\delta_2(\epsilon)$.

Example 4.28. Let $\varphi(x)=\frac{x^2}{1-x}$ be the studied rational function and consider now the closed interval $I'=[x_L,x_U]\subset[0,1)$. We want to find δ_2 such that for every $x\in[x_L, x_U-\delta_2]$ and $0<\delta<\delta_2$ it holds

$$|\varphi(x+\delta)-\varphi(x)|<\epsilon.$$

The rational function φ is strictly increasing on the interval $[0, 1)$. Since $x < x+\delta$, the absolute value in the inequality above can be removed. By evaluating the function φ we achieve

$$\frac{x^2 + 2\delta x + \delta^2}{1 - x - \delta} - \frac{x^2}{1 - x} < \epsilon.$$

This inequality can be modified as

$$\delta^2(1 - x) + \delta \left(-x^2 + x(2 - \epsilon) + \epsilon\right) - \epsilon(x - 1)^2 < 0.$$

We have $0 < \delta < 1 - x$ and $0 \leq x < 1$. The accuracy $\epsilon > 0$ is fixed. Using the quadratic formula, we can conclude that the inequality above holds for

$$0 < \delta < \frac{x^2 - x(2 - \epsilon) - \epsilon + \sqrt{\left(-x^2 + x(2 - \epsilon) + \epsilon\right)^2 - 4\epsilon(x - 1)^3}}{2(1 - x)}.$$

Thus, the value δ_2 can be chosen as the minimal value of the expression on the right hand side for $x \in [x_L, x_U]$, i.e.

$$\delta_2(\epsilon) := \min_{x_L \leq x \leq x_U} \left(\frac{x^2 - x(2 - \epsilon) - \epsilon + \sqrt{\left(-x^2 + x(2 - \epsilon) + \epsilon\right)^2 - 4\epsilon(x - 1)^3}}{2(1 - x)} \right).$$

Let us apply this formula on the situation from Example 4.24. After five while-loops we were in the situation $x_L = \frac{15}{16}, x_U = \frac{31}{32}$. Recall that $\epsilon = 1$. Thus, we can evaluate the function δ_2 as

$$\delta_2(1) = \min_{\frac{15}{16} \leq x \leq \frac{31}{32}} \left(\frac{x^2 - x - 1 + \sqrt{\left(-x^2 + x + 1\right)^2 - 4(x - 1)^3}}{2(1 - x)} \right).$$

The expression is strictly increasing on the interval $[\frac{15}{16}, \frac{31}{32}]$, therefore

$$\delta_2(1) = \left(\frac{\frac{31}{32}^2 - \frac{31}{32} - 1 + \sqrt{\left(-\frac{31}{32}^2 + \frac{31}{32} + 1\right)^2 - 4(\frac{31}{32} - 1)^3}}{2(1 - \frac{31}{32})} \right) \doteq 0.00094784$$

Since $L = 22$, we can conclude that we need at most

$$\lfloor \log_2(22) - \log_2(\delta_2(1)) \rfloor + 2 = 14 + 2 = 16$$

while-loops to halt.

We see that expressing δ_2 as a function of ϵ is non-trivial even in this simple example. Also note, that the function $\delta_2(\epsilon)$ is generally not continuous. However, it is independent of the input value y. Thus, it can be understood as a machine constant.

4.5 Overview

Table 4.3 summarizes what we know about the problem of inverting a rational function φ which is continuous on I, where I is either an open interval or a point.

Table 4.3: Inverting a rational function φ on an open interval or a point I

	φ	$J = \varphi(I)$	$\epsilon = 0$	$\epsilon > 0$	
				Constant time	Computable
1.	linear	UB	✓	✓	✓
2.	non-linear	P	✓	✓	✓
3.		B	✗	✓	✓
4.		UB	✗	(✗)	✓

The abbreviation “UB” stands for “unbounded” which means that J is an unbounded open interval. The abbreviation “B” stands for “bounded” which means that J is a bounded open interval. And finally “P” stands for “point” and it expresses that J is a point.

The meaning of the symbols “✓” and “✗” will be explained in the discussion below in which we describe each cell of Table 4.3 separately.

- The column “$\epsilon = 0$” expresses the inversion with infinite precision.
  - The first row consider a rational function which can be represented by a fraction of two linear polynomials. We discussed at the beginning of Chapter 3 that such a function can be inverted with infinite precision in constant time on any interval.
  - We discussed at the beginning of Section 4.4 that if the component J is a point then we can invert the function φ on I with infinite precision in constant time.
  - In Corollary 3.2 we proved that the function $\varphi(x) = x^2$ cannot be inverted with infinite precision by any BSS machine. Similarly we could prove that the same holds for any rational function which cannot be expressed as a fraction of two linear polynomials. This explains the crosses in the remaining rows.
- The column “$\epsilon > 0$ + Constant time” refers to the inversion with fixed precision $\epsilon > 0$ by a constant time BSS machine.
  - Rows 1 and 2 follow from the previous column.
  - From Observation 4.25 we know that the machine M which we constructed in Section 4.4 is a constant time BSS machine if the rational function φ and the open interval I are such that the interval $J = \varphi(I)$ is bounded.
  - In Theorem 3.3 we proved that x^2 cannot be inverted by a constant time BSS machine on an unbounded interval. It is not difficult to prove that the same holds for every function of the form $\varphi(x) = x^n$,

  where $|n| \geq 2$ (the idea of the proof is similar as in the case $n = 2$). Therefore, we conjecture that no rational function which satisfies the conditions of this row is invertible by a constant time BSS machine.

- In Section 4.4 we proved that any rational function which is continuous on an open interval I can be inverted with an arbitrary but fixed precision $\epsilon > 0$. This explains all the positive results in the last column; i.e. the column labelled "$\epsilon > 0$ + Computable".

In this chapter we defined easy to compute real functions as functions that are computable by a constant time BSS machine. Consequently, if ψ is an easy to compute real function then it can be defined by finitely many rational functions. We proved that every easy to compute function ψ is invertible upto some fixed precision $\epsilon > 0$. Moreover, if the real function ψ consists (only) of rational functions that are bounded on corresponding intervals then the function ψ is invertible in constant time. Furthermore, we showed how to calculate an upper bound on the time complexity of the inversion problem for a concrete easy to compute real function ψ.

5. Further remarks

In this chapter we bring up two interesting topics regarding hard to invert functions in the BSS model of computation which would be worth further study.

5.1 Inverting an arbitrary BSS machine over $\mathbb{R}$

In the previous chapter we defined that a real function $\psi\colon \mathbb{R} \to \mathbb{R}$ is easy to compute in the BSS model of computation if it is an input-output map of a *constant* time BSS machine M_0. What happens is we omit the time restriction of the machine M_0 in the definition?

First of all note that the inverting machine M which we constructed in the previous chapter would generally not invert the machine M_0. By the Path Decomposition Theorem the halting set of a time unbounded BSS machine is a *countable* disjoint union of semi-algebraic sets. Consequently, to fully describe the decomposition of the set Ω_{M_0} we need infinitely many values. And recall that the set of machine constants is a finite set.

We do not see why this alternative definition of an easy to compute function should bring some new results in the question of hard to invert functions. In other words, that there is a computable real function whose inverse is not computable while considering inversion with precision $\epsilon > 0$.

However, let us suggest two BSS machines M_{BER} and M_{LOG} that would be interesting to analyse.

The first machine is based on the *Bernoulli map* which is well known due to its chaotic behaviour. It iterates the function $\text{BER}\colon [0,1] \to [0,1]$ defined as

$$\text{BER}(x) = \begin{cases} 2 \cdot x, & \text{if } x \leq \frac{1}{2}, \\ 2 \cdot x - 1, & \text{if } x > \frac{1}{2}. \end{cases}$$

It is easy to observe that the BSS machine which k-times iterates the function BER has a piecewise linear input-output map and that the halting set of such a machine decomposes in 2^k open intervals and $2^k + 1$ points. See the book [12] for more details about the Bernoulli map.

Let us briefly discuss how the BSS machine M_{BER} defined in Figure 5.1 works. The machine has a small constant $\delta > 0$. On an input x from the interval $[0,1]$ it applies iteratively the Bernoulli function until it reaches the the interval $[0,\delta]$. Then it outputs the number of performed iterations.

An adversary which is trying to invert this machine is given a natural number $k \in \mathbb{N}$. His task is to find $x' \in [0,1]$ such that $\Phi_{M_{\text{BER}}}(x') = k$. Note that in this case it does not matter if we require inversion with infinite precision or inversion with the fixed precision ϵ, where $0 \leq \epsilon < \frac{1}{2}$.

Another interesting machine, which would be worth further analysis, uses the previous machine in combination with another chaotic function. Namely the *logistic map* which iterates the function $\text{LOG}\colon [0,1] \to [0,1]$ defined as

$$\text{LOG}(x) = r \cdot x \cdot (1 - x),$$

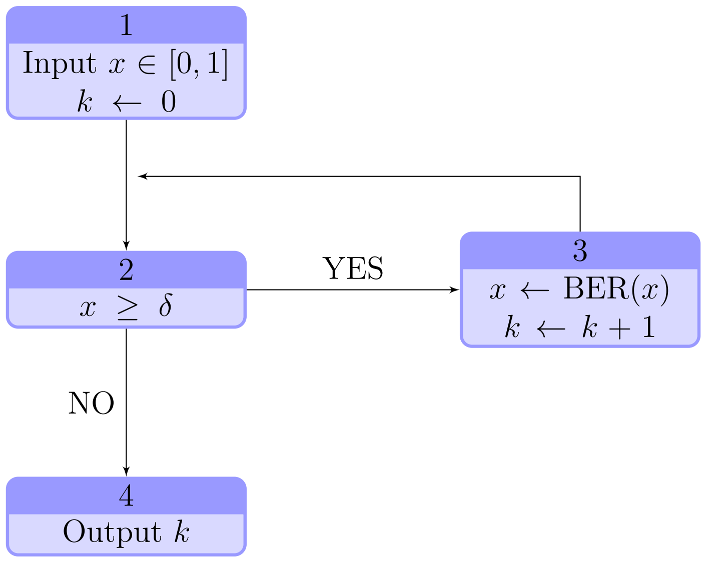


Figure 5.1: The graph of a BSS machine M_{BER} which is based on the Bernoulli map

where $r \in (0,4]$ is some fixed constant. A lot of research have been done in order to analyse the behaviour of the logistic map for different choices of the constant r. According to the author of the book [12] the logistic map is chaotic for most of the values r from the interval $(3.57, 4)$. It is not difficult to observe that the BSS machine which k-times iterates the function LOG has only one halting path, i.e. the halting set consists of one connected component $[0,1]$. On the other hand, the input-output map is a polynomial function of degree 2^k.

Figure 5.2 describes a machine M_{LOG} which uses both functions BER and LOG. On an input $x \in [0,1]$ it first runs the machine M_{BER}. The output of the machine M_{BER} determines the number of iterations of the function LOG applied on the input x.

An adversary is now given a real number $y \in [0,1]$. His task is to find $x' \in [0,1]$ such that $|\Phi_{M_{\text{LOG}}}(x') - y| < \epsilon$ for some fixed precision $\epsilon > 0$. Note that in this case inversion with infinite precision does not make sense since the input-output map is not piecewise linear (see Section 3.2 for more details).

5.2 Can Turing machines simulate general BSS machines over $\mathbb{R}$?

In this section we briefly explain why we think that Turing machines cannot simulate BSS machines over real numbers even if restricted to integer inputs.

Assume that f is a length-preserving permutation

$$\begin{gathered} f\colon \{0,1\}^* \to \{0,1\}^*, \\ x \in \{0,1\}^n \mapsto y \in \{0,1\}^n, \end{gathered}$$

which is computable by a polynomial time Turning machine (or equivalently, by a general BSS machine over $\mathbb{Z}_2$, recall Section 2.4). We can think of f as being

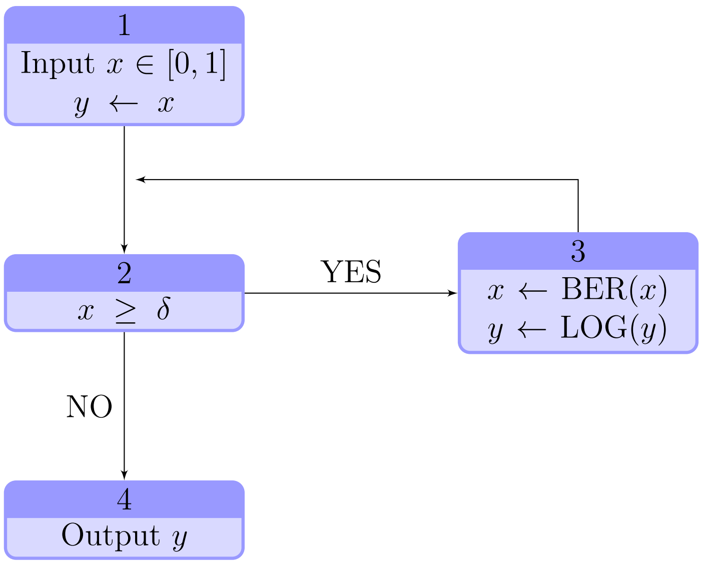


Figure 5.2: The graph of a BSS machine M_{LOG} which is based on two chaotic maps; namely, the Bernoulli map and the logistic map

defined on natural numbers and represent it by the following real function

$$\begin{aligned}\widetilde{f}\colon \mathbb{R}^+ &\to \mathbb{N},\\ x &\mapsto f(\lfloor x \rfloor).\end{aligned}$$

It can be observed that the time complexity of a general BSS machine over $\mathbb{R}$ computing $\widetilde{f}$ is polynomial in n, where $n = \lceil \log(x+1) \rceil$. Recall the end of Subsection 2.1.7, where we discussed the time complexity of a BSS machine computing the floor function.

Let M be a general BSS machine over $\mathbb{R}$ which inverts the function $\widetilde{f}$. That is, given a value $y = \widetilde{f}(x)$ it finds a real number $x' \in \mathbb{R}^+$ such that

$$\widetilde{f}(x') = y.$$

In other words, the machine M can approximate the value x with the precision $\frac{1}{2}$. Suppose that the time complexity of the BSS machine M is polynomial in n, where $n = \lceil \log(x+1) \rceil$. Our question is the following.

Would the existence of the BSS machine M from above imply the existence of a polynomial time Turing machine inverting the permutation f?

The BSS machine M is a machine over $\mathbb{R}$; thus, it can have finitely many real numbers as machine constants. Every real number can by its digits encode a subset of $\mathbb{R}$. Consequently, we can understand constants of machine M as an oracle access.

This is why we conjecture that Turing machines cannot simulate BSS machines over $\mathbb{R}$, even just on integer inputs.

Conclusion

In this thesis we primarily studied real functions computable by constant time BSS machines with one dimensional input space and one dimensional output space. Our main theme was to analyse how difficult it is to invert these functions.

In order to do so, we firstly had to specify what we mean by "inverting" a function in the BSS model of computation. In this work we took into consideration two possible approaches. In the first variant of the definition we required inversion with infinite precision whereas in the second one we allowed inversion with some fixed precision $\epsilon > 0$.

We proved that even very simple real functions, such as for example x^2, are not invertible with respect to the first definition of inversion. Consequently, there are infinitely many hard to invert real functions in the framework of this definition.

On the other hand, we showed that there is no real function which is easy to compute but impossible to invert in the sense of the second variant of the definition. Our proof was constructive which means that we provided a method how to find an inverting machine for any given easy to compute real function. This approach allowed us to do a detailed time complexity analysis of the inverting machines. As a result, we were able to discuss which real functions are invertible in constant time.

An example of a real function which is invertible with fixed precision $\epsilon > 0$ but not in constant time is x^2. For this concrete function we presented a more efficient inverting machine than for a general function and we calculated its time complexity. In addition, we derived a formula for a lower bound on the time complexity of the problem of inverting x^2. We expressed the lower bound explicitly under the assumption that 0 and 1 are the only constants of the inverting machine.

There are several ways how to extend the work done in this thesis. It would be interesting to study how difficult it is to invert an arbitrary real function which is computable in the BSS model of computation. We presented two BSS machines whose inverse would be worth further analysis. Both suggested machines are based on chaotic maps. Another direction building up on this work is to consider real functions with n dimensional input space and subsequently also real maps.

At the very end of this thesis we briefly discussed why we agree with the commonly recognized conjecture that general BSS machines over $\mathbb{R}$ are more powerful objects than Turing machines.

Bibliography

[1] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation.* Springer-Verlag New York, Inc., 1998. ISBN: 978-0-387-98281-6.

[2] Lenore Blum, Michael Shub, and Steve Smale. On a Theory of Computation and Complexity Over the Real Numbers: NP-Completeness, Recursive Functions and Universal Machines. *Bulletin of Amererican Mathematical Society (N.S.)*, 21(1):1–46, 1989.

[3] Mark Braverman and Stephen A. Cook. Computing over the Reals: Foundations for Scientific Computing. *Notices of the American Mathematical Society*, 53(3):318–329, 2006.

[4] Peter Bürgisser and Peter Scheiblechner. On the complexity of counting components of algebraic varieties. *Journal of Symbolic Computation*, 44(9):1114 – 1136, 2009. Effective Methods in Algebraic Geometry.

[5] Felipe Cucker. $\mathrm{P}_{\mathbb{R}} \neq \mathrm{NC}_{\mathbb{R}}$. *Journal of Complexity*, 8(3):230–238, 1992.

[6] David S. Dummit and Richard M. Foote. *Abstract algebra.* John Wiley & Sons, Inc., Hoboken, NJ, third edition, 2004. ISBN: 978-0-471-43334-7.

[7] Dima Grigoriev and Sergey I. Nikolenko. Continuous hard-to-invert functions and biometric authentication. *Groups – Complexity – Cryptology*, 4(1):19–32, 2012.

[8] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994. ISBN: 978-0-201-53082-7.

[9] Walter Rudin. *Principles of mathematical analysis.* International Series in Pure and Applied Mathematics. McGraw-Hill Book Co., New York, third edition, 1976. ISBN: 978-0070542358.

[10] Michael Sipser. *Introduction to the Theory of Computation.* Cengage Learning, Boston, third edition, 2012. ISBN: 978-1-133-18779-0.

[11] Steve Smale. Newton's Method Estimates from Data at One Point. In Richard E. Ewing, Kenneth I. Gross, and Clyde F. Martin, editors, *The Merging of Disciplines: New Directions in Pure, Applied, and Computational Mathematics*. Springer-Verlag, 1986. ISBN: 978-1-4612-4984-9.

[12] Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering.* Addison-Wesley, 1994. ISBN: 978-0-201-54344-5.

[13] L. Van den Dries. *Tame Topology and O-minimal Structures*, volume 248 of *London Mathematical Society Lecutre Note Series.* Cambridge University Press, 1998. ISBN: 978-0-521-59838-5.

[14] Xiaodong Wang. Some results relevant to Smale's reports. In Morris W. Hirsch, Jerrold E. Marsden, and Michael Shub, editors, *From Topology to Computation: Proceedings of the Smalefest*, pages 456–465. Springer-Verlag, 1993. ISBN: 978-1-4612-2740-3.

[15] David P. Woodruff and Marten van Dijk. Cryptography in an unbounded computational model. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, chapter 10, pages 149–164. Springer Berlin Heidelberg, 2002. ISBN: 978-3-540-46035-0.

List of Figures

List of Tables

List of Algorithms