Exact Arithmetic on the Stern-Brocot Tree

M. Niqui

Nijmegen Institute for Computing and Information Sciences/

NIII-R0325 November 2003

Nijmegen Institute for Computing and Information Sciences
Faculty of Science
Catholic University of Nijmegen
Toernooiveld 1
6525 ED Nijmegen
The Netherlands

Exact Arithmetic on the Stern–Brocot Tree

Milad Niqui
University Of Nijmegen
Department of Computer Science
milad@cs.kun.nl

Abstract

In this paper we present the Stern–Brocot tree as a basis for performing exact arithmetic on rational and real numbers. We introduce the tree and mention its relation with continued fractions. Based on the tree we present a binary representation of rational numbers and investigate various algorithms to perform exact rational arithmetic using a simplified version of the homographic and the quadratic algorithms [19, 12]. We show generalisations of homographic and quadratic algorithms to multilinear forms in n variables and we prove the correctness of the algorithms. Finally we modify the tree to get a redundant representation for real numbers.

Keywords: *Stern–Brocot tree*, *exact arithmetic*, *homographic*, *quadratic*, *multilinear*.
Classification: 11A55, 11A67, 68-04 (AMS'00); I.1.2, G.1.0, F.2.1 (CR'98).

1 History

Recently exact arithmetic has been considered as a suitable approach to the problem of dealing with round-off errors and building more reliable and versatile programming tools for computation with rational and real numbers. According to this approach, real numbers are represented as an infinite stream over a finite or infinite alphabet and the computation over them is done in a *lazy* manner[1]: in order to compute a function on a real number, we start absorbing the first element of the stream representing the real number. At each step we output an element of the output stream or we may need more information about the input, in which case we absorb the next element of the input stream.

There have been many theoretical and practical instances of applying this idea. A rather general theoretical approach is taken by Konečný [8], where the limitation theorems for IFS-representations of real numbers is given. IFS-representations are representations of real numbers using an infinite composition of contracting functions on a compact interval. These include the representations by means of Möbius transformations that is developed by Potts and Edalat [15, 4]. Potts and Edalat's work generalises earlier works by Gosper [5], Vuillemin [19] and Menissier-Morain [12]. Gosper, in his famous unpublished work [5], showed how to add and multiply two continued fractions. He introduced the idea of using homographic and quadratic algorithms. His algorithms are presented for regular $\mathbb{N}$-fractions of rational numbers and were the first instance of exact arithmetic on rational numbers. Later Vuillemin [19] adapted the algorithms to work for a redundant representation of real numbers using continued fractions. Kornerup and Matula [9] presented a binary version of Gosper's algorithm on a bit-serial arithmetic unit. Their binary encoding of continued fraction expansion was a lexicographic one and they could use it to obtain a redundant representation for real numbers [10]. Our approach in the present paper is similar to the one by Kornerup and Matula. We introduce a binary representation of rational numbers based on continued fractions and we expand it to a redundant representation for real numbers.

Our basis is the Stern–Brocot tree which was first discovered by 19th century German mathematician Moritz Abraham Stern and French clockmaker Achille Brocot [3, 17, 7]. The tree itself

[1] Also called: on-line, call-by-need, corecursive, coalgebraic, etc..

is a symmetric mathematical structure with remarkable algebraic and combinatorial properties. It was recently reintroduced by Graham et al. [6]. Bates [1] studies the tree thoroughly and compares it with other similar combinatorial structures such as Farey sequence, hyperbinary tree, Gray-code sequence and paper folding sequence. Both in [6] and in [1] a binary representation for positive rational numbers based on the tree is introduced. This representation basically boils down to the unary encoding of the regular continued fraction expansion of rational numbers and is presented in Raney [16]. Raney uses this binary representation to devise the homographic algorithm on continued fractions and he makes use of finite-state automata called *transducers*. Liardet and Stambul [11] present the quadratic algorithm using Raney's transducers and generalise it to compute rational functions involving continued fraction expansions. Bertot [2] discusses the fact that by taking this binary encoding for rational numbers one can simplify the complexity of mathematical proofs.

In Section 2 we introduce the Stern–Brocot tree and some of its basic properties. We also consider the binary representation obtained from the tree. In Section 3 we show the relation between Stern–Brocot representation and continued fraction expansion. In Section 4 we show the exact rational arithmetic on the binary Stern–Brocot representation. This is done by presenting homographic algorithms (Section 4.1) and quadratic algorithms (Section 4.2). There we also generalise these algorithms to the case of more than two variables to get multilinear algorithms in Section 4.3. In that section we also present a proof of correctness of these class of algorithms. In Section 5 we modify the Stern–Brocot tree to get a ternary redundant representation which is suitable for computation with real numbers.

2 Introduction

The Stern–Brocot tree is the full binary tree in which all nodes are labelled in such a way that each positive rational number occurs exactly once. The tree is 'hanging on a rope between zero and infinity'. Projecting the tree vertically will provide the usual ordering of the rationals. For $\frac{p_1}{q_1}$ and $\frac{p_2}{q_2}$ we define the *mediant* of them to be the fraction $\frac{p_1+p_2}{q_1+q_2}$. We refer to $\frac{p_1}{q_1}$ and $\frac{p_2}{q_2}$ as *parents* of $\frac{p_1+p_2}{q_1+q_2}$. In this tree every row consists of the fractions that are mediants of elements of previous rows. We start to write the two pseudo-fractions $\frac{0}{1}$ and $\frac{1}{0}$, we proceed to construct the tree row by row. The first row is the mediant of the two initial pseudo-fractions, that is $\frac{0+1}{1+0} = \frac{1}{1}$. We write this mediant in the middle of the initial pseudo-fractions. The second row consists of the mediant of $(\frac{0}{1}, \frac{1}{1})$ and the mediant of $(\frac{1}{1}, \frac{1}{0})$. We continue in this way and each time we choose two fractions which don't have anything vertically between them and we place their mediant in a new row and vertically in the middle of its two parents. This construction is illustrated in Figure I. The resulting tree has many interesting properties. We mention some of the basic properties here. For a proof of this lemma and more properties including some combinatorial properties see [6, 1].

Lemma 2.1

i) All the fractions occurring in the Stern–Brocot tree are irreducible.

ii) Every positive rational number occurs exactly once in the Stern–Brocot tree.

Proof. See [6, pp 117-119]. □

From Lemma 2.1.ii it follows that the Stern–Brocot tree is an enumeration of the positive rational numbers. It has already been observed [6] that this property can be used to give a binary encoding of positive rational numbers. We start from the root of the tree and we represent $\frac{1}{1}$ by the empty sequence. To encode an arbitrary fraction, we find it in the tree and consider the path from the root to that fraction. We obtain the encoding by starting from the empty sequence in the root and adding **L** each time we go to left and adding **R** each time we go to right. For example the representation of $\frac{5}{12}$ is **LLRRL**. Conversely, given a finite sequence of **R**s and **L**s we can use it to locate the corresponding fraction in the tree, simply by starting from the root

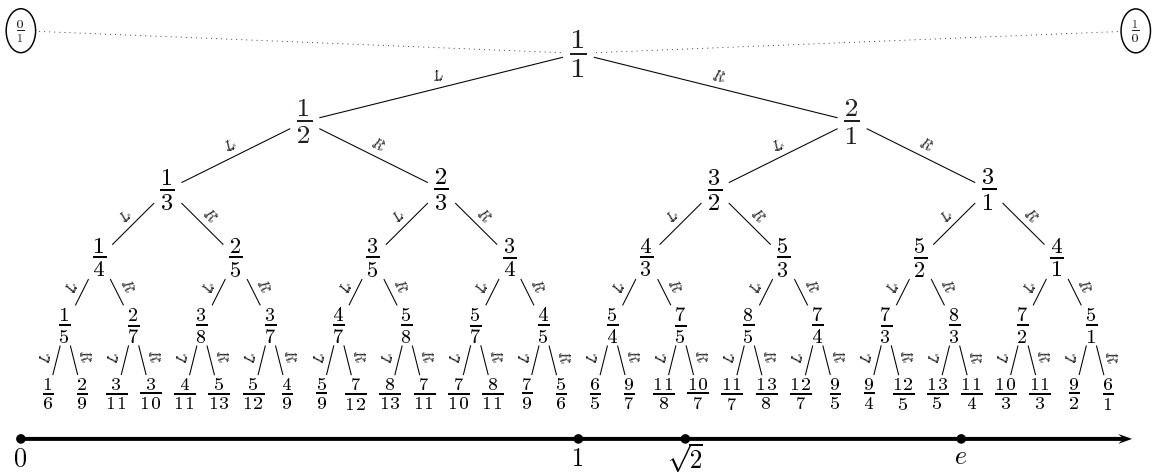

Figure I: The Stern–Brocot tree

and going to the left each time we encounter an **L** and going to the right each time we encounter an **R**. For example the fraction corresponding to **RLRLR** is $\frac{13}{8}$. We call this *the Stern–Brocot representation* of positive rational numbers and throughout the rest of the paper we try to give a computer friendly treatment of this specific representation. First we fix some notations. Let $\Sigma = \alpha_0, \alpha_1, \ldots, \alpha_n$ be a finite set. By Σ^* we denote the set of finite strings of elements of Σ. By Σ^ω we denote the set of of infinite streams of elements of Σ. The nonempty set Δ is a *representation* if it is a subset of $\Sigma^* \cup \Sigma^\omega$ for some Σ. Let Δ be a representation. Let $\delta, \delta' \in \Delta$, if δ is finite then $\delta\delta'$ is the string concatenation of δ and δ'. By $\delta|_n$ we mean the string consisting of the first n elements of δ and $\delta(n)$ means the nth element of δ. We say that δ' is an initial segment of δ and denote it by $\delta' \sqsubseteq \delta$ if there exists $\delta'' \in \Delta$ s.t. $\delta = \delta'\delta''$. If $\alpha \in \Sigma$ is an alphabet and $\delta \in \Delta$ is an string, then $\alpha\delta$ denotes the string obtained by prepending α to the beginning of δ. We denote the empty string by [].

Next we show how to find the Stern–Brocot representation of a positive fraction without traversing the Stern–Brocot tree.

Definition 2.2 Let $\mathbf{SB} = \{\mathbf{L}, \mathbf{R}\}^*$ be the set of finite strings generated from two letters **L** and **R**. We define the following injection from the set of positive rational numbers $\mathbb{Q}^+$ to **SB**:

$$\begin{cases} \ulcorner \_ \urcorner : \mathbb{Q}^+ \longmapsto \mathbf{SB} \\ \ulcorner \frac{m}{n} \urcorner := \begin{cases} [] & ; m = n, \\ \mathbf{L} \ulcorner \frac{m}{n-m} \urcorner & ; m < n, \\ \mathbf{R} \ulcorner \frac{m-n}{n} \urcorner & ; m > n. \end{cases} \end{cases}$$

For every $x \in \mathbb{Q}^+$ we call $\ulcorner x \urcorner \in \mathbf{SB}$ *the unsigned Stern–Brocot binary representation of* x or in short *the SB-encoding of* x.

Note that we do not require $\gcd(m, n) = 1$. As a corollary to Lemma 2.1 one can prove that $\ulcorner \frac{m}{n} \urcorner = \ulcorner \frac{km}{kn} \urcorner$. Moreover the following important property holds which links the Stern–Brocot tree and the Stern–Brocot representation:

Lemma 2.3 *The outcome of the function* $\ulcorner \_ \urcorner$ *is the same as traversing the Stern–Brocot tree and following the left and right branches at each step.*

Proof. See [6, pp 120-122]. □

Next we define the inverse map that given any binary sequence in **SB** returns the corresponding rational number.

Definition 2.4 Let $\sigma \in \mathbf{SB}$. We define the map $[\![\_]\!] : \mathbf{SB} \longrightarrow \mathbb{Q}^+$ as the following:

$$\begin{cases} [\![[]]\!] & := 1 \\ [\![\mathbf{L}\sigma]\!] & := \frac{[\![\sigma]\!]}{[\![\sigma]\!]+1} \\ [\![\mathbf{R}\sigma]\!] & := [\![\sigma]\!] + 1 \end{cases}$$

The following lemma tells us that $\ulcorner \_ \urcorner$ and $[\![\_]\!]$ are inverse to each other and hence we have a bijection between positive rational numbers and finite binary sequences. The proof follows from the definition by a simple case analysis.

Lemma 2.5 ($\ulcorner \_ \urcorner$ is a bijection.) *The function $[\![\_]\!]$ is an injection from **SB** to $\mathbb{Q}^+$ and moreover is the inverse of $\ulcorner \_ \urcorner$:*

$$\forall q \in \mathbb{Q}^+ \quad [\![\ulcorner q \urcorner]\!] = q, \quad \forall \sigma \in \textbf{\textit{SB}} \ \ulcorner [\![\sigma]\!] \urcorner = \sigma.$$

It is easy to equip the set **SB** with a sign bit to get the entire $\mathbb{Q}$. We define the following data type:

$$\mathbf{SSB} = \mathbf{ZERO} \mid \mathbf{POS}\ \ \mathbf{SB} \mid \mathbf{NEG}\ \ \mathbf{SB}$$

We call this new set *the signed Stern–Brocot representation.*

Definition 2.6 We extend the $\ulcorner \_ \urcorner$ and $[\![\_]\!]$ functions to the entire $\mathbb{Q}$ and **SSB**. For example if $q \in \mathbb{Q}$ by $\ulcorner q \urcorner$ we mean the image of q in **SSB**:

$$\ulcorner q \urcorner := \begin{cases} \mathbf{ZERO} & q = 0, \\ \mathbf{POS}\ \ulcorner q \urcorner & 0 < q, \\ \mathbf{NEG}\ \ulcorner (-q) \urcorner & q < 0. \end{cases}$$

Thus the set **SSB** is isomorphic to $\mathbb{Q}$. Now we are interested in transferring the computations on $\mathbb{Q}$ to computations on **SSB**. In other words we seek a way of equipping the set **SSB** with algebraic operations and relations such that they are more efficient than usual addition and multiplication on $\mathbb{Q}$ as a set of pairs of integers. One important property of **SSB** is that it is inductively defined, so we can define functions over **SSB** by recursion. Note that we do not use quotients in defining **SSB**, as we do in the usual definition of $\mathbb{Q}$ as set of pairs of integers. Thus, our goal is to define, by recursion, suitable addition and multiplication over **SSB**. We will do this in Section 4. After that one might want to extend the set of finite sequences to contain all infinite binary streams and consider the set $\mathbf{SSB}^\omega = \{\mathbf{L}, \mathbf{R}\}^\omega$. As expected, this set coincides with the set of real numbers and corresponds to the infinite branches in the Stern–Brocot tree. But this representation is not computationally suitable. We will investigate this in more details in Section 5.

3 Stern–Brocot Tree and continued fractions

There is an intrinsic connection between the Stern–Brocot representation of a rational number $\frac{m}{n}$ and Euclid's algorithm applied to m, n. By slightly changing the definition of $\ulcorner \frac{m}{n} \urcorner$ we can get a new function which calculates the $\gcd(m, n)$.

$$\begin{cases} \gcd'(\_,\_) : \mathbb{N} \times \mathbb{N} \longmapsto \mathbb{N} \\ \gcd'(m,n) := \begin{cases} m & m = n, \\ \gcd'(m, n-m) & m < n, \\ \gcd'(m-n, n) & m > n. \end{cases} \end{cases}$$

This is the usual Euclid's algorithm but instead of division (in fact instead of the mod function) we only use subtraction. This analogy subsequently relates the Stern–Brocot and continued fraction representation of a rational number:

Definition 3.1 (ℕ-fraction) Let $p, q \in \mathbb{Z}$ and $q \neq 0$. Then $[a_0, a_1, \ldots, a_n]$ is the ℕ-*fraction* of $\frac{p}{q}$, if using Euclid's algorithm we can find r_i such that:

$$\begin{array}{lll} p & = a_0 q + r_0 & 0 < r_0 < q \\ q & = a_1 r_0 + r_1 & 0 < r_1 < r_0 \\ r_0 & = a_2 r_1 + r_2 & 0 < r_2 < r_1 \\ \ldots & & \\ r_{n-2} & = a_n r_{n-1} + 0, & \end{array}$$

and $r_{n-1} = \gcd(p, q)$.

It can be easily observed that in the ℕ-fraction of any rational number, the first element is an integer and the rest of the elements are positive integers. The following lemma presents the connection between Stern–Brocot representation and continued fractions.

Lemma 3.2 *Let $X = \mathbf{R}^{i_0}\mathbf{L}^{j_0}\mathbf{R}^{i_1}\mathbf{L}^{j_1}\cdots\mathbf{R}^{i_n}\mathbf{L}^{j_n}$ be the binary Stern–Brocot representation of the* positive *rational number x. Then we have:*

$$\mathbb{N}\textit{-fraction of } x = \begin{cases} [i_0, j_0, i_1, j_1, \cdots, i_n, j_n + 1] & j_n > 0, \\ [i_0, j_0, i_1, j_1, \cdots, i_n + 1] & j_n = 0. \end{cases}$$

Proof. See [6]. □

From this lemma it follow that there is a bijection from **SB** to the set of ℕ-fractions of positive fractions.

The relationship between the SB-encoding and Euclid's algorithm can be seen as a justification of the main property of Stern–Brocot tree expressed in Lemma2.1. More specifically, by inspecting the above gcd′ function, the SB-encoding can be seen as keeping track of the number of times numerator (resp. denominator) is larger than the denominator (resp. numerator) by outputting **L** (resp. **R**) during the evaluation the greatest common divisor of numerator and denominator. This in turn can be seen as coding the proof of the fact that the fraction is irreducible. In other words irreducible fractions correspond to canonical proofs of the fact they are irreducible. This analogy is explained in details by Bertot in [2].

4 Exact Rational Arithmetic

In this section we present the algorithms for computation with the Stern–Brocot representation. According to Lemma. 3.2, there is a close relationship between continued fraction arithmetic and Stern–Brocot representation. Inspired from algorithms for arithmetic on continued fractions we devise our new algorithms directly on **SSB**. The main difference is that here we refrain from using Euclid's division and only use subtraction. First we define the homographic algorithm and then quadratic algorithm. As in the case of continued fractions, each algorithm has two main steps. We call them *absorption* and *emission* steps. In the absorption steps we ask for the first bit of input(s). In the emission step we output one bit. The idea is to ask for bits of input and produce bits of output as soon as possible. This is a typical lazy algorithm in which the program may print the correct answer without absorbing all the terms of the input(s). The reason why this is possible is simply the continuity (in our case for positive input) of the class of multilinear transformation (cf. Section 4.3) which means by getting each bit of input we can approximate the output; and if our approximation is precise enough — which in this case depends on the four integer coefficients—, then we can firmly determine the 'next' element of output. We continue in this way until we eventually run out of input or arrive at a stage where the rest of output is independent of *any additional* input. This method was proposed by Gosper [5] for arithmetic on continued fractions but is easily applicable to a wide variety of problems that deal with computing a continued function, that is to say almost all algorithms in numerical analysis [15].

4.1 Homographic algorithm

A *homographic transformation* on rational numbers is a function of the form:

$$h_M(x) = \frac{ax+b}{cx+d} \qquad a,b,c,d \in \mathbb{Z}; \qquad M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

Throughout the rest of the paper we implicitly require that the denominators of all fractions are nonzero. We never explicitly mention it even as a precondition to the algorithms. When there is no risk of ambiguity, we omit the subscript that refers to the matrix of coefficients.

In this section our goal is to give an algorithm to compute the Stern–Brocot representation of $\ulcorner h([\![\sigma]\!]) \urcorner$ for $\sigma \in \mathbf{SSB}$. Thus we are given a finite signed binary sequence and four integer coefficients and we want to produce a finite signed binary sequence which is the Stern–Brocot representation of $h(x)$.

In order to present the algorithm we only need to assume $\sigma \in \mathbf{SB}$, i.e. σ is the Stern–Brocot representation of a *positive* rational number. The algorithm that we obtain to compute $\ulcorner h([\![\sigma]\!]) \urcorner$ can be easily extended to an algorithm for computing $\ulcorner h([\![\tau]\!]) \urcorner$ when $\tau \in \mathbf{SSB}$. This is simply because if $[\![\tau]\!] = -[\![\sigma]\!]$ then:

$$\ulcorner h_M([\![\tau]\!]) \urcorner = \ulcorner \frac{a[\![\tau]\!]+b}{c[\![\tau]\!]+d} \urcorner = \ulcorner \frac{(-a)[\![\sigma]\!]+b}{(-c)[\![\sigma]\!]+d} \urcorner = \ulcorner h_{M'}([\![\sigma]\!]) \urcorner.$$

Hence we let $\sigma \in \mathbf{SB}$. First we determine the sign of $h_M([\![\sigma]\!])$. In the course of determining the sign, each time we absorb input the coefficient matrix changes. Thus the algorithm that determines the sign should not only output the sign but also the new coefficient matrix and the remaining (i.e. not yet absorbed) part of the input. Next step is to devise an algorithm that given the sign, the new coefficients and the remaining part of the input calculates the bits of the output. We break this step into two simpler steps

- We devise an algorithm that assumes the numerator and the denominator are positive and calculates the output bits. We call this the *output-bits* algorithm.
- We combine the sign algorithm and the output-bits algorithm to compute $h_M([\![\sigma]\!])$ for $\sigma \in \mathbf{SB}$.

In the rest of this section we abbreviate the coefficient matrix $\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$ by M and we use the following conventions: $M_N := \left[\begin{smallmatrix} a-c & b-d \\ c & d \end{smallmatrix}\right]$, $M_D := \left[\begin{smallmatrix} a & b \\ c-a & d-b \end{smallmatrix}\right]$, $M_{\mathbf{L}} := \left[\begin{smallmatrix} a+b & b \\ c+d & d \end{smallmatrix}\right]$ and $M_{\mathbf{R}} := \left[\begin{smallmatrix} a & a+b \\ c & c+d \end{smallmatrix}\right]$. To compute the sign note that $0 < [\![\sigma]\!]$, so all we need to do is to check the sign of the two lines $ax+b$ and $cx+d$ for positive x; in other words we have to check the sign of $\frac{-b}{a}$ and $\frac{-d}{c}$. To avoid integer division, we only need to check the value of $\operatorname{sgn} a + \operatorname{sgn} b$ and $\operatorname{sgn} c + \operatorname{sgn} d$. Thus we define the following function:

Definition 4.1 Let $m, n \in \mathbb{Z}$. We define the *sum of sign of m and n* to be $f_1(m,n) = \operatorname{sgn} m + \operatorname{sgn} n$.

Lemma 4.2

i) $f_1(m,n) > 0$ *then* $\forall x > 0\ mx+n > 0$.

ii) $f_1(m,n) < 0$ *then* $\forall x > 0\ mx+n < 0$.

iii) $f_1(f_1(a,b), f_1(c,d)) = 2$ *or* $f_1(f_1(a,b), f_1(c,d)) = -2$ *then* $\forall x > 0\ \frac{ax+b}{cx+d} > 0$.

iv) $f_1(f_1(a,b), f_1(c,d)) = 0$ *then* $\forall x > 0\ \frac{ax+b}{cx+d} < 0$.

Proof. This is a corollary to Lemma 4.14 which will be proved in Section 4.3 □

The sign algorithm is given below. The input is a matrix of coefficients $M = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$ and an Stern–Brocot binary sequence $\sigma \in \mathbf{SB}$. The output is a triple $(s, \left[\begin{smallmatrix} a_s & b_s \\ c_s & d_s \end{smallmatrix}\right], \sigma_s)$ consisting of the sign, the new coefficients matrix and the remaining input. The algorithm is written in the form of a recursive function $\mathcal{S}_{\_} : M_{2\times 2}(\mathbb{Z}) \times \mathbf{SB} \longmapsto \{0, +1, -1\} \times M_{2\times 2}(\mathbb{Z}) \times \mathbf{SB}$:

Sign algorithm for $a, b, c, d \in \mathbb{Z}, \sigma \in \boldsymbol{SB}$.

$$
\begin{cases}
\mathcal{S}_M([]) := \begin{cases} (0, M, []) & \textbf{if } \operatorname{sgn} a + \operatorname{sgn} b = 0, \\ (+1, M, []) & \textbf{else if } \operatorname{sgn} a + \operatorname{sgn} b = \operatorname{sgn} c + \operatorname{sgn} d, \\ (-1, M, []) & \textbf{otherwise}. \end{cases} \\
\mathcal{S}_M(\mathbf{L}\sigma) := \begin{cases} \textbf{if } b = 0 & \begin{cases} ((\operatorname{sgn} a)(\operatorname{sgn} c), M, \sigma) & \textbf{if } d = 0, \\ (\operatorname{sgn} a, M, \sigma) & \textbf{else if } f_1(c,d) > 0, \\ (-\operatorname{sgn} a, M, \sigma) & \textbf{else if } f_1(c,d) < 0, \\ \mathcal{S}_{M_{\mathbf{L}}}(\sigma) & \textbf{otherwise}. \end{cases} \\ \textbf{if } b \neq 0 & \begin{cases} \textbf{if } d = 0 & \begin{cases} (\operatorname{sgn} c, M, \sigma) & \textbf{if } f_1(a,b) > 0, \\ (-\operatorname{sgn} c, M, \sigma) & \textbf{else if } f_1(a,b) < 0, \\ \mathcal{S}_{M_{\mathbf{L}}}(\sigma) & \textbf{otherwise}. \end{cases} \\ \textbf{if } d \neq 0 & \begin{cases} (+1, M, \sigma) & \textbf{if } f_1(f_1(a,b), f_1(c,d)) = \pm 2, \\ (-1, M, \sigma) & \textbf{else if } f_1(f_1(a,b), f_1(c,d)) = 0, \\ \mathcal{S}_{M_{\mathbf{L}}}(\sigma) & \textbf{otherwise}. \end{cases} \end{cases} \end{cases} \\
\mathcal{S}_M(\mathbf{R}\sigma) := \begin{cases} \textbf{if } b = 0 & \begin{cases} ((\operatorname{sgn} a)(\operatorname{sgn} c), M, \sigma) & \textbf{if } d = 0, \\ (\operatorname{sgn} a, M, \sigma) & \textbf{else if } f_1(c,d) > 0, \\ (-\operatorname{sgn} a, M, \sigma) & \textbf{else if } f_1(c,d) < 0, \\ \mathcal{S}_{M_{\mathbf{R}}}(\sigma) & \textbf{otherwise}. \end{cases} \\ \textbf{if } b \neq 0 & \begin{cases} \textbf{if } d = 0 & \begin{cases} (\operatorname{sgn} c, M, \sigma) & \textbf{if } f_1(a,b) > 0, \\ (-\operatorname{sgn} c, M, \sigma) & \textbf{else if } f_1(a,b) < 0, \\ \mathcal{S}_{M_{\mathbf{R}}}(\sigma) & \textbf{otherwise}. \end{cases} \\ \textbf{if } d \neq 0 & \begin{cases} (+1, M, \sigma) & \textbf{if } f_1(f_1(a,b), f_1(c,d)) = \pm 2, \\ (-1, M, \sigma) & \textbf{else if } f_1(f_1(a,b), f_1(c,d)) = 0, \\ \mathcal{S}_{M_{\mathbf{R}}}(\sigma) & \textbf{otherwise}. \end{cases} \end{cases} \end{cases}
\end{cases}
$$

For the output-bit algorithm we assume that we are given a matrix of coefficients $\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$ and an Stern–Brocot binary sequence $\sigma \in \mathbf{SB}$. Moreover we need the precondition that if $\sigma = []$ then $a + b > 0$ and $c + d > 0$. This is because in this algorithm in case $\sigma = []$ we are going to evaluate the Stern–Brocot representation of $\frac{a+b}{c+d}$.

Definition 4.3 Let $m, n, p, q \in \mathbb{Z}$. We define the predicate

$$\psi_1(m, n, p, q) := (p \leq m \wedge q < n) \vee (p < m \wedge q \leq n).$$

We use these trivial properties in our algorithms:

Lemma 4.4

i) If $\psi_1(m, n, p, q)$ *then* $\forall x > 0\ mx + n > px + q$.

ii) If $\neg\psi_1(m, n, p, q) \wedge \neg\psi_1(p, q, m, n)$ *then* $\exists x > 0\ mx + n = px + q$.

The output-bit algorithm is given below. The input is a matrix of coefficients $M = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$ and a Stern–Brocot binary sequence $\sigma \in \mathbf{SB}$. The output is a Stern–Brocot binary sequence $\sigma \in \mathbf{SB}$.

The algorithm is written in the form of a recursive function $\mathcal{B}_{\_} : M_{2\times 2}(\mathbb{Z}) \times \mathbf{SB} \longmapsto \mathbf{SB}$:

output-bit algorithm for $a, b, c, d \in \mathbb{Z}, \sigma \in \boldsymbol{SB}$.

$$\begin{cases} \mathcal{B}_M([]) := \ulcorner \dfrac{a+b}{c+d} \urcorner \\ \mathcal{B}_M(\mathbf{L}\sigma) := \begin{cases} \mathbf{R}\ (\mathcal{B}_{M_N}(\mathbf{L}\sigma)) & \textbf{if } \psi_1(a,b,c,d), \\ \mathbf{L}\ (\mathcal{B}_{M_D}(\mathbf{L}\sigma)) & \textbf{else if } \psi_1(c,d,a,b), \\ \mathcal{B}_{M_{\mathbf{L}}}(\sigma) & \textbf{otherwise}. \end{cases} \\ \mathcal{B}_M(\mathbf{R}\sigma) := \begin{cases} \mathbf{R}\ (\mathcal{B}_{M_N}(\mathbf{R}\sigma)) & \textbf{if } \psi_1(a,b,c,d), \\ \mathbf{L}\ (\mathcal{B}_{M_D}(\mathbf{R}\sigma)) & \textbf{else if } \psi_1(c,d,a,b), \\ \mathcal{B}_{M_{\mathbf{R}}}(\sigma) & \textbf{otherwise}. \end{cases} \end{cases}$$

At this point we have to combine the above algorithms and obtain the homographic function. The input is a matrix of coefficients $M = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$ and a Stern–Brocot binary sequence $\sigma \in \mathbf{SB}$. The output is a signed Stern–Brocot binary sequence. First we calculate $\mathcal{S}_M(\sigma)$ and we assume that the result is (s, M_s, σ_s) where $M_s = \left[\begin{smallmatrix} a_s & b_s \\ c_s & d_s \end{smallmatrix}\right]$. Furthermore we use the following conventions: $M_s^- = \left[\begin{smallmatrix} -a_s & -b_s \\ -c_s & -d_s \end{smallmatrix}\right]$, $M_{s_1}^- = \left[\begin{smallmatrix} a_s & b_s \\ -c_s & -d_s \end{smallmatrix}\right]$ and $M_{s_2}^- = \left[\begin{smallmatrix} -a_s & -b_s \\ c_s & d_s \end{smallmatrix}\right]$. The algorithm is written in the form of a function $\mathcal{H}_{\_} : M_{2\times 2}(\mathbb{Z}) \times \mathbf{SB} \longmapsto \mathbf{SSB}$. Note that there is no recursive calls in this function:

homographic algorithm for $a, b, c, d \in \mathbb{Z}, \sigma \in \boldsymbol{SB}$.

$$\begin{cases} \mathcal{H}_M(\sigma) := \begin{cases} \textbf{if } ad = bc & \begin{cases} \ulcorner \frac{a}{c} \urcorner & \textbf{if } c \neq 0, \\ \ulcorner \frac{b}{d} \urcorner & \textbf{otherwise}. \end{cases} \\ \textbf{if } ad \neq bc & \begin{cases} \mathbf{ZERO} & \textbf{if } s = 0, \\ \textbf{if } s = 1 & \begin{cases} \mathbf{POS}\ \mathcal{B}_{M_s}(\sigma_s) & \textbf{if } a_s + b_s > 0, \\ \mathbf{POS}\ \mathcal{B}_{M_s^-}(\sigma_s) & \textbf{otherwise}. \end{cases} \\ \textbf{if } s = -1 & \begin{cases} \mathbf{NEG}\ \mathcal{B}_{M_{s_1}^-}(\sigma_s) & \textbf{if } a_s + b_s > 0, \\ \mathbf{NEG}\ \mathcal{B}_{M_{s_2}^-}(\sigma_s) & \textbf{otherwise}. \end{cases} \end{cases} \end{cases} \end{cases}$$

The following theorem shows the correctness of the algorithms above. The proof of this theorem will be given (in a rather general form) in Section 4.3:

Theorem 4.5 *Let* $h_M(x) = \frac{ax+b}{cx+d}$ *where* $M = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right] \in M_{2\times 2}(\mathbb{Z})$. *Let* $\sigma \in \boldsymbol{SB}$.

i) Assume $\mathcal{S}_M(\sigma) = (s, M_s, \sigma_s)$. *Then* $s = \operatorname{sgn} h_M([\![\sigma]\!])$.

ii) If $a + b > 0$ *and* $c + d > 0$ *then* $\ulcorner h_M([\![\sigma]\!]) \urcorner = \mathcal{B}_M(\sigma)$.

iii) $\ulcorner h_M([\![\sigma]\!]) \urcorner = \mathcal{H}_M(\sigma)$

4.2 Quadratic algorithm

In this section we give a synchronised algorithm to compute addition and multiplication of two **SB**-encodings. To do this we consider the following general quadratic transformation:

$$q_A(x, y) = \frac{axy + bx + cy + d}{exy + fx + gy + h} \qquad a, b, c, d, e, f, g, h \in \mathbb{Z} \qquad \text{and } T = \begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix}.$$

We denote the set of $2 \times (2 \times 2)$ tensors with integer elements by $T_{2\times(2\times 2)}(\mathbb{Z})$. One could consider a 2×4 matrix instead of a tensor. We use the term tensor to be consistent with the existing literature.

By considering the quadratic functions, we have a much more general algorithm and by taking special tensors for T we obtain the algorithms for basic arithmetic operations.

$$T_{\oplus} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{\otimes} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$T_{\ominus} = \begin{bmatrix} 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{\oslash} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

And we define arithmetic operations on **SSB** as:

$$\begin{cases} \oplus : \mathbf{SSB} \times \mathbf{SSB} \longmapsto \mathbf{SSB}, \\ \sigma_1 \oplus \sigma_2 := \ulcorner q_{T_\oplus} \urcorner([\![\sigma_1]\!], [\![\sigma_2]\!]). \end{cases} \quad \begin{cases} \otimes : \mathbf{SSB} \times \mathbf{SSB} \longmapsto \mathbf{SSB}, \\ \sigma_1 \otimes \sigma_2 := \ulcorner q_{T_\otimes} \urcorner([\![\sigma_1]\!], [\![\sigma_2]\!]). \end{cases}$$

$$\begin{cases} \ominus : \mathbf{SSB} \times \mathbf{SSB} \longmapsto \mathbf{SSB}, \\ \sigma_1 \ominus \sigma_2 := \ulcorner q_{T_\ominus} \urcorner([\![\sigma_1]\!], [\![\sigma_2]\!]). \end{cases} \quad \begin{cases} \oslash : \mathbf{SSB} \times \mathbf{SSB} \longmapsto \mathbf{SSB}, \\ \sigma_1 \oslash \sigma_2 := \ulcorner q_{T_\oslash} \urcorner([\![\sigma_1]\!], [\![\sigma_2]\!]). \end{cases}$$

This algorithm is very similar to the homographic algorithm of the previous section. Once again it suffices to assume $\sigma_1, \sigma_2 \in \mathbf{SB}$, i.e. σ_1 and σ_2 are the Stern–Brocot representation of *positive* rational numbers. The algorithm we obtain to compute $\ulcorner q([\![\sigma_1]\!], [\![\sigma_2]\!]) \urcorner$ is easily extendible to an algorithm for computing $\ulcorner q([\![\tau_1]\!], [\![\tau_2]\!]) \urcorner$ when $\tau_1, \tau_2 \in \mathbf{SSB}$. There are three different cases to consider according to whether one or both of τ_1, τ_2 being negative[2].

Hence we assume $\sigma_1, \sigma_2 \in \mathbf{SB}$ are given and we proceed exactly the same way as the in previous section. In the rest of this section we denote by T the coefficient tensor $\left[\begin{smallmatrix} a & b & c & d \\ e & f & g & h \end{smallmatrix}\right]$ and we use the following convention for the modified coefficient matrices after absorption of inputs or emission of output:

$$T_{\mathbf{LL}} = \left[\begin{smallmatrix} a+b+c+d & b+d & c+d & d \\ e+f+g+h & f+h & g+h & h \end{smallmatrix}\right] \quad T_{\mathbf{LR}} = \left[\begin{smallmatrix} a+c & a+b+c+d & c & c+d \\ e+g & e+f+g+h & g & g+h \end{smallmatrix}\right] \quad T_{\mathbf{RL}} = \left[\begin{smallmatrix} a+b & b & a+b+c+d & b+d \\ e+f & f & e+f+g+h & f+h \end{smallmatrix}\right]$$
$$T_{\mathbf{RR}} = \left[\begin{smallmatrix} a & a+b & a+c & a+b+c+d \\ e & e+f & e+g & e+f+g+h \end{smallmatrix}\right] \quad T_N = \left[\begin{smallmatrix} a-e & b-f & c-g & d-h \\ e & f & g & h \end{smallmatrix}\right] \quad T_D = \left[\begin{smallmatrix} a & b & c & d \\ e-a & f-b & g-c & h-d \end{smallmatrix}\right].$$

First we determine the sign of $q_T([\![\sigma_1]\!], [\![\sigma_2]\!])$. In contrast with the sign algorithm for the homographic function we absorb the bits from *both* inputs. The algorithm that determines the sign should not only output the sign but also the new coefficients matrix and the remaining part of both inputs. Then we present the output-bit algorithm that for a positive numerator and denominator emits the outputs.Finally the quadratic algorithm combines the sign algorithm and the output-bits algorithm to compute $q_T([\![\sigma_1]\!], [\![\sigma_2]\!])$ for $\sigma_1, \sigma_2 \in \mathbf{SB}$. First we need a definition similar to the Definition 4.1:

Definition 4.6 Let $m, n, p, q \in \mathbb{Z}$. We define the *sum of sign of m,n,p and q* to be $f_2(m, n) := \operatorname{sgn} m + \operatorname{sgn} n + \operatorname{sgn} p + \operatorname{sgn} q$.

Lemma 4.7

i) $f_2(m, n, p, q) > 2$ *then* $\forall x, y > 0\ mxy + nx + py + q > 0$.

ii) $f_2(m, n, p, q) < -2$ *then* $\forall x, y > 0\ mxy + nx + py + q < 0$.

iii) $f_1(f_2(a, b, c, d) - 2, f_2(e, f, g, h) - 2) = 2$ *or* $f_1(f_2(a, b, c, d) + 2, f_2(e, f, g, h) + 2) = -2$ *then* $\forall x, y > 0\ \frac{axy+bx+cy+d}{exy+fx+gy+h} > 0$.

iv) $f_1(f_2(a, b, c, d) - 2, f_2(e, f, g, h) + 2) = 0$ *or* $f_1(f_2(a, b, c, d) + 2, f_2(e, f, g, h) - 2) = 0$ *then* $\forall x, y > 0\ \frac{axy+bx+cy+d}{exy+fx+gy+h} < 0$.

[2] For example if $[\![\tau_1]\!] = -[\![\sigma_1]\!]$ and $[\![\tau_2]\!] = [\![\sigma_2]\!]$ then:

$$\ulcorner q_T([\![\tau_1]\!], \tau_2) \urcorner = \ulcorner \frac{a[\![\tau_1]\!][\![\tau_2]\!] + b[\![\tau_1]\!] + c[\![\tau_2]\!] + d}{e[\![\tau_1]\!][\![\tau_2]\!] + f[\![\tau_1]\!] + g[\![\tau_2]\!] + h} \urcorner = \ulcorner \frac{(-a)[\![\sigma_1]\!][\![\sigma_2]\!] - b[\![\sigma_1]\!] + c[\![\sigma_2]\!] + d}{(-e)[\![\sigma_1]\!][\![\sigma_2]\!] - f[\![\sigma_1]\!] + g[\![\sigma_2]\!] + h} \urcorner = \ulcorner q_{T'}([\![\sigma_1]\!], [\![\sigma_2]\!]) \urcorner.$$

Proof. This is a corollary to Lemma 4.14 which will be proved in Section 4.3 □

The sign algorithm is given below. The input is a tensor of coefficients $T = \left[\begin{smallmatrix} a & b & c & d \\ e & f & g & h \end{smallmatrix}\right]$ and two Stern–Brocot binary sequence $\sigma_1, \sigma_2 \in \mathbf{SB}$. The output is a quadruple $(s, T_s, \sigma_{1_s}, \sigma_{2_s})$ consisting of the sign, the new coefficient tensor and the remaining parts of both inputs. The algorithm is written in the form of a recursive function $\mathcal{S}_{2\_} : T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB} \longmapsto \{0, +1, -1\} \times T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB}$.

In the following let $A_1 = \left[\begin{smallmatrix} a+b & c+d \\ e+f & g+h \end{smallmatrix}\right]$, $A_2 = \left[\begin{smallmatrix} a+c & b+d \\ e+g & f+h \end{smallmatrix}\right]$, $S_{A_1}(\sigma_1) = (s_1, B_{s_1}, \sigma_{s_1})$, $S_{A_2}(\sigma_2) = (s_2, B_{s_2}, \sigma_{s_2})$, $B_{s_1} = \left[\begin{smallmatrix} a_{s_1} & b_{s_1} \\ c_{s_1} & d_{s_1} \end{smallmatrix}\right]$ and $B_{s_2} = \left[\begin{smallmatrix} a_{s_2} & b_{s_2} \\ c_{s_2} & d_{s_2} \end{smallmatrix}\right]$. Let $T_{B_{s_1}} = \left[\begin{smallmatrix} 0 & a_{s_1} & 0 & b_{s_1} \\ 0 & c_{s_1} & 0 & d_{s_1} \end{smallmatrix}\right]$ and $T_{B_{s_2}} = \left[\begin{smallmatrix} 0 & 0 & a_{s_2} & b_{s_2} \\ 0 & 0 & c_{s_2} & d_{s_2} \end{smallmatrix}\right]$. Let $\kappa_1 = f_2(a, b, c, d)$ and $\kappa_2 = f_2(e, f, g, h)$:

Sign algorithm for $a, b, c, d, e, f, g, h \in \mathbb{Z}, \sigma_1, \sigma_2 \in \boldsymbol{SB}$.

$$
\begin{cases}
\mathcal{S}_{2T}(\sigma_1, []) := (s_1, T_{B_{s_1}}, \sigma_{s_1}, []) \\
\mathcal{S}_{2T}([], \sigma_2) := (s_2, T_{B_{s_2}}, [], \sigma_{s_2}) \\
\mathcal{S}_{2T}(\sigma_1, \sigma_2) := \\
\quad \begin{cases}
\textbf{if } b = c = d = 0 & \begin{cases}
((\operatorname{sgn} a)(\operatorname{sgn} e), T, \sigma_1, \sigma_2) & \textbf{if } f = g = h = 0, \\
(\operatorname{sgn} a, T, \sigma_1, \sigma_2) & \textbf{else if } \kappa_2 > 2, \\
(-\operatorname{sgn} a, T, \sigma_1, \sigma_2) & \textbf{else if } \kappa_2 < -2, \\
\mathcal{X}_T(\sigma_1, \sigma_2) & \textbf{otherwise}.
\end{cases} \\
\textbf{otherwise} & \begin{cases}
\textbf{if } f = g = h = 0 & \begin{cases}
(\operatorname{sgn} e, T, \sigma_1, \sigma_2) & \textbf{else if } \kappa_1 > 2, \\
(-\operatorname{sgn} e, T, \sigma_1, \sigma_2) & \textbf{else if } \kappa_1 < -2, \\
\mathcal{X}_T(\sigma_1, \sigma_2) & \textbf{otherwise}.
\end{cases} \\
\textbf{otherwise} & \begin{cases}
(+1, T, \sigma_1, \sigma_2) & \textbf{if } f_1(\kappa_1 - 2, \kappa_2 - 2) = 2 \\
 & \vee f_1(\kappa_1 + 2, \kappa_2 + 2) = -2, \\
(-1, T, \sigma_1, \sigma_2) & \textbf{else if } f_1(\kappa_1 - 2, \kappa_2 + 2) = 0 \\
 & \vee f_1(\kappa_1 + 2, \kappa_2 - 2) = 0, \\
\mathcal{X}_T(\sigma_1, \sigma_2) & \textbf{otherwise}.
\end{cases}
\end{cases}
\end{cases}
\end{cases}
$$

The function $\mathcal{X}_{\_} : T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB} \longmapsto \{0, +1, -1\} \times T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB}$ is defined by pattern matching on the head element of input. This function is actually the recursive step in the $\mathcal{S}_{2\_}$ function and is also the step in which we absorb the head elements of inputs. Note that at each call to this function we absorb the head element of *both* inputs simultaneously[3]:

$$
\begin{cases}
\mathcal{X}_T(\mathbf{L}\sigma_1', \mathbf{L}\sigma_2') := \mathcal{S}_{2T_{\mathbf{LL}}}(\sigma_1', \sigma_2') \\
\mathcal{X}_T(\mathbf{L}\sigma_1', \mathbf{R}\sigma_2') := \mathcal{S}_{2T_{\mathbf{LR}}}(\sigma_1', \sigma_2') \\
\mathcal{X}_T(\mathbf{R}\sigma_1', \mathbf{L}\sigma_2') := \mathcal{S}_{2T_{\mathbf{RL}}}(\sigma_1', \sigma_2') \\
\mathcal{X}_T(\mathbf{R}\sigma_1', \mathbf{R}\sigma_2') := \mathcal{S}_{2T_{\mathbf{RR}}}(\sigma_1', \sigma_2')
\end{cases}
$$

Next we give the output-bit algorithm. Assume we are given a tensor T of coefficients and a Stern–Brocot binary sequence $\sigma \in \mathbf{SB}$. Moreover we need the precondition that if one of σ_1 or σ_2 is the empty sequence then $a{+}b{+}c{+}d{>}0$ and $e{+}f{+}g{+}h{>}0$. This is because in this algorithm we are going to evaluate the function $\mathcal{B}$ of the previous section, which had a similar precondition. First we need a generalisation of the Definition 4.3:

[3] There are other absorption strategies possible but we don't consider this issue in this paper. For a discussion of possible absorption strategies we refer the reader to [15, Ch. 11].

Definition 4.8 Let $m,n,p,q,r,s,t,u \in \mathbb{Z}$. We define the predicate:

$$\begin{aligned}\psi_2(m,n,p,q,r,s,t,u) :=& (r \leq m \wedge s \leq n \wedge t \leq p \wedge u < q) \vee (r \leq m \wedge s \leq n \wedge t < p \wedge u \leq q) \\ & \vee (r \leq m \wedge s < n \wedge t \leq p \wedge u \leq q) \vee (r < m \wedge s \leq n \wedge t \leq p \wedge u \leq q)\end{aligned}$$

Lemma 4.9

i) If $\psi_2(m,n,p,q,r,s,t,u)$ *then* $\forall x,y > 0\ mxy + nx + py + q > rxy + sx + ty + u$.

ii) If $\neg\psi_2(m,n,p,q,r,s,t,u) \wedge \neg\psi_2(m,n,p,q,r,s,t,u)$ *then*

$$\exists x,y > 0\ mxy + nx + py + q = rxy + sx + ty + u.$$

We are ready to present the output-bit algorithm. The algorithm is given in the form of a recursive function $\mathcal{B}_{2\_} : T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB} \longmapsto \mathbf{SB}$:

output-bit algorithm for $a,b,c,d,e,f,g,h \in \mathbb{Z}, \sigma_1,\sigma_2 \in \boldsymbol{SB}$.

$$\begin{cases} \mathcal{B}_{2T}(\sigma_1,[]) := \mathcal{B}_{A_1}(\sigma_1) \\ \mathcal{B}_{2T}([],\sigma_2) := \mathcal{B}_{A_2}(\sigma_2) \\ \mathcal{B}_{2T}(\sigma_1,\sigma_2) := \begin{cases} \mathbf{R}\ (\mathcal{B}_{2T_N}(\sigma_1,\sigma_2)) & \textbf{if } \psi_2(a,b,c,d,e,f,g,h), \\ \mathbf{L}\ (\mathcal{B}_{2T_D}(\sigma_1,\sigma_2)) & \textbf{else if } \psi_2(e,f,g,h,a,b,c,d), \\ \mathcal{X}'_T(\sigma_1,\sigma_2) & \textbf{otherwise}. \end{cases} \end{cases}$$

The function $\mathcal{X}'_{\_} : T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB} \longmapsto \mathbf{SB}$ is defined by pattern matching on the head element of input. This function is in fact the recursive step in the $\mathcal{B}_{2\_}$ function where we perform the recursion on the input sequences:

$$\begin{cases} \mathcal{X}'_T(\mathbf{L}\sigma'_1,\mathbf{L}\sigma'_2) := \mathcal{B}_{2T_{\mathbf{LL}}}(\sigma'_1,\sigma'_2) \\ \mathcal{X}'_T(\mathbf{L}\sigma'_1,\mathbf{R}\sigma'_2) := \mathcal{B}_{2T_{\mathbf{LR}}}(\sigma'_1,\sigma'_2) \\ \mathcal{X}'_T(\mathbf{R}\sigma'_1,\mathbf{L}\sigma'_2) := \mathcal{B}_{2T_{\mathbf{RL}}}(\sigma'_1,\sigma'_2) \\ \mathcal{X}'_T(\mathbf{R}\sigma'_1,\mathbf{R}\sigma'_2) := \mathcal{B}_{2T_{\mathbf{RR}}}(\sigma'_1,\sigma'_2) \end{cases}$$

Next we have to combine the above algorithms and obtain the quadratic function. The input is a tensor of coefficients and two Stern–Brocot binary sequences $\sigma_1,\sigma_2 \in \mathbf{SB}$. The output is a signed Stern–Brocot binary sequence. First we calculate $\mathcal{S}_{2T}(\sigma_1,\sigma_2)$ and we assume that the result is $(s,T_s,\sigma_{1_s},\sigma_{2_s})$ where $T_s = \begin{bmatrix} a_s & b_s & c_s & d_s \\ e_s & f_s & g_s & h_s \end{bmatrix}$. Furthermore we use the following conventions: $T_s^- = \begin{bmatrix} -a_s & -b_s & -c_s & -d_s \\ -e_s & -f_s & -g_s & -h_s \end{bmatrix}$, $T_{s_1}^- = \begin{bmatrix} a_s & b_s & c_s & d_s \\ -e_s & -f_s & -g_s & -h_s \end{bmatrix}$ and $T_{s_2}^- = \begin{bmatrix} -a_s & -b_s & -c_s & -d_s \\ e_s & f_s & g_s & h_s \end{bmatrix}$. We also need to define a predicate which generalises the notion of singularity for non square matrices (in our case the $2 \times (2 \times 2)$ tensor):

Definition 4.10 For $T = \begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix}$ we define the predicate *same_ratio* (T) as:

$$\textit{same\_ratio}\ (T) := af = be \wedge bg = cf \wedge ch = dg \wedge ag = ce \wedge ah = de \wedge bh = df.$$

The algorithm is written in the form of a function $\mathcal{Q}_{\_} : T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB} \longmapsto \mathbf{SSB}$. Note that there is no recursive calls in this function:

quadratic algorithm for $a,b,c,d,e,f,g,h \in \mathbb{Z}, \sigma_1, \sigma_2 \in \boldsymbol{SB}$.

$$
\left\{\begin{array}{l}
\\
\\
\\
\\
\\
\\
\\
\\
\end{array}\right.
\mathcal{Q}_T(\sigma_1,\sigma_2) :=
\left\{\begin{array}{ll}
\textbf{if } \textit{same\_ratio}\ (T) & \left\{\begin{array}{ll}
\ulcorner \frac{d}{h} \urcorner & \textbf{if } h \neq 0, \\
\ulcorner \frac{c}{g} \urcorner & \textbf{if } g \neq 0, \\
\ulcorner \frac{b}{f} \urcorner & \textbf{if } f \neq 0, \\
\ulcorner \frac{a}{e} \urcorner & \textbf{otherwise}.
\end{array}\right. \\
\\
\textbf{otherwise} & \left\{\begin{array}{ll}
\textbf{ZERO} & \textbf{if } s = 0, \\
\textbf{if } s = 1 & \left\{\begin{array}{ll}
\textbf{POS}\ \mathcal{B}_{2T_s}(\sigma_{1_s},\sigma_{2_s}) & \textbf{if } a_s+b_s+c_s+d_s > 0, \\
\textbf{POS}\ \mathcal{B}_{2T_s^-}(\sigma_{1_s},\sigma_{2_s}) & \textbf{otherwise}.
\end{array}\right. \\
\textbf{if } s = -1 & \left\{\begin{array}{ll}
\textbf{NEG}\ \mathcal{B}_{2T_{s_1}^-}(\sigma_{1_s},\sigma_{2_s}) & \textbf{if } a_s+b_s+c_s+d_s > 0, \\
\textbf{NEG}\ \mathcal{B}_{2T_{s_2}^-}(\sigma_{1_s},\sigma_{2_s}) & \textbf{otherwise}.
\end{array}\right.
\end{array}\right.
\end{array}\right.
$$

To conclude this section we mention the following theorem which shows the correctness of the above algorithms. This will be a corollary to the Theorem 4.21 of the next section:

Theorem 4.11 *Let* $q_T(x) = \frac{axy+bx+cy+d}{exy+fx+gy+h}$ *where* $T = \left[\begin{smallmatrix} a & b & c & d \\ e & f & g & h \end{smallmatrix}\right] \in T_{2\times(2\times 2)}(\mathbb{Z})$. *Let* $\sigma_1, \sigma_2 \in \boldsymbol{SB}$.

i) Assume $\mathcal{S}_{2T}(\sigma_1,\sigma_2) = (s, T_s, \sigma_{1_s}, \sigma_{2_s})$. *Then* $s = \operatorname{sgn} q_T([\![\sigma_1]\!],[\![\sigma_2]\!])$.

ii) If $a+b+c+d > 0$ *and* $e+f+g+h > 0$ *then* $\ulcorner q_T([\![\sigma_1]\!],[\![\sigma_2]\!])\urcorner = \mathcal{B}_{2T}(\sigma_1,\sigma_2)$.

iii) $\ulcorner q_T([\![\sigma_1]\!],[\![\sigma_2]\!])\urcorner = \mathcal{Q}_T(\sigma_1,\sigma_2)$

4.3 Multilinear Forms

Looking back at the algorithms of the previous two sections, we can observe that the quadratic algorithms are obtained by generalising the homographic algorithms to two variables. The passage from functions $\mathcal{S}_M$ to $\mathcal{S}_{2T}$, from $\mathcal{B}_M$ to $\mathcal{B}_{2T}$ and from $\mathcal{H}_M$ to $\mathcal{Q}_T$ is straightforward. Another generalisation, perhaps less clear, is the way the homographic algorithm is a generalised form of the $\ulcorner\_\urcorner$ function of Section 2.

The main purpose of the present section is to give a unified proof of the Theorems 4.5 and 4.11. In order to do that we introduce the class of multilinear functions of n variables. Next we explore a general form of the aforementioned algorithms which given a list of n Stern–Brocot binary sequences, compute a multilinear fractional form of n variables. This generalised algorithm is not necessarily very efficient as we only pay attention to the mathematical properties of the functions and sometimes we assume that certain subroutines are given without explicitly mentioning them.

Definition 4.12 We define the family $\mathfrak{L}_n$ of *multilinear polynomials of n variables* recursively as follows:

1. $\mathfrak{L}_0 = \mathbb{Z}$.

2. $\mathfrak{L}_{n+1}(x_1,\ldots,x_n,x_{n+1}) = x_{n+1}\mathfrak{L}_n(x_1,\ldots,x_n) + \mathfrak{L}_n(x_1,\ldots,x_n)$

If $L_n, L'_n \in \mathfrak{L}_n$ we call $\mu_n = \frac{L_n}{L'_n}$ a *multilinear fractional form of n variables.*

In other words a multilinear polynomial of n variables is a polynomial that is linear with respect to each subset of its arguments. As an example,

$$L_3(x,y,z) = axyz + bxy + cyz + dxz + ex + fy + gz + h$$

is a member of $\mathfrak{L}_3$. Note that rational numbers are multilinear fractional forms of 0 variables, and homographic and quadratic functions are multilinear fractional forms of respectively 1 and 2 variables.

To generalise the quadratic algorithm we consider the following form:

$$\mu_n(x_1,\ldots,x_n)=\frac{L_n(x_1,\ldots,x_n)}{L'_n(x_1,\ldots,x_n)},\qquad L_n,L'_n\in\mathfrak{L}_n.$$

Throughout the rest of the section we assume that $a_0,\ldots,a_{2^n}$ are coefficients of $L_n(x_1,\ldots,x_n)$ in ascending order (i.e. a_0 is the constant and a_{2^n} is the coefficient of the monomial with n variables). Similarly we show the coefficients of $L'_n(x_1,\ldots,x_n)$ by $b_0,\ldots,b_{2^n}$. This way we can arrange the coefficients in a 2×2^n tensor:

$$T_n=\begin{bmatrix} a_{2^n} & a_{2^n-1} & \ldots & a_0\\ b_{2^n} & b_{2^n-1} & \ldots & b_0\end{bmatrix}$$

Instead of considering the tensor as an $n+1$-dimensional cube, we consider it as a rectangular 2×2^n matrix. We explicitly mention the tensor whenever we have a multilinear fractional form. Next we want to present the algorithm that given $\sigma_1,\cdots,\sigma_n\in\mathbf{SB}$ calculates the function $\ulcorner\mu_{n_{T_n}}([\![\sigma_1]\!],\ldots,[\![\sigma_n]\!])\urcorner$. In order to be able to do that we need to fix some conventions.

We present some notations for the tensors of modified coefficients whenever we absorb from input or emit an output bit. For absorbing we always follow the simultaneous absorption strategy, which means that we absorb the head bit of all n input sequences at the same step. This way we have 2^n different cases with respect to the head bit of each of n input sequences. In order to encode this we consider the lexicographic ordering on the n element binary sequences of **L**s and **R**s. We show this by the sequence $\pi_1,\ldots,\pi_{2^n}$. For example:

$$\pi_1=\underbrace{\mathbf{L}\mathbf{L}\cdots\mathbf{L}}_{n\ times},\quad \pi_2=\underbrace{\mathbf{L}\mathbf{L}\cdots\mathbf{L}}_{n-1\ times}\mathbf{R},\quad \pi_{2^{n-3}+1}=\mathbf{L}\mathbf{L}\mathbf{L}\mathbf{R}\underbrace{\mathbf{L}\cdots\mathbf{L}}_{n-4\ times}.$$

As an example in the case of quadratic function, we have 4 absorption tensors which were mentioned in Section 4.2 as $T_{\mathbf{LL}}$, $T_{\mathbf{LR}}$, $T_{\mathbf{RL}}$ and $T_{\mathbf{RR}}$. By tensor T_{π_i} we mean the tensor of coefficients after absorbing the head elements of the n input sequences, where those absorbed elements are $\pi_i(1),\pi_i(2),\ldots,\pi_i(n)$. For example T_{π_1} denotes the tensor of modified coefficients where all the inputs start with **L**. We use the transformations given in Definition 2.4 to calculate the new coefficients. If we define $\phi(\mathbf{L},x)=\frac{x}{x+1}$ and $\phi(\mathbf{R},x)=x+1$, one can see that if $\sigma_k=\pi_i(k)\sigma'_k$ (i.e. if σ' is the tail of σ); then $\phi(\pi_i(k),[\![\sigma'_k]\!])=[\![\sigma_k]\!]$ and:

$$\begin{aligned}\mu_{n_{T_n}}([\![\sigma_1]\!],\cdots,[\![\sigma_n]\!]) &= \mu_{n_{T_n}}(\phi(\pi_i(1),[\![\sigma'_1]\!]),\cdots,\phi(\pi_i(n),[\![\sigma'_n]\!]))\\ &= \mu_{n_{T_{\pi_i}}}([\![\sigma'_1]\!],\cdots,[\![\sigma'_n]\!]).\end{aligned}\tag{4.12.1}$$

This equation shows that given π_i and T_n, one can always calculate T_{π_i}. In this section we do not give an explicit algorithm for this, but we assume that such an algorithm is given. It is obvious that in the case of homographic and quadratic algorithm no algorithm is necessary as we are only dealing with 2 (resp. 4) tensors.

As a next step we present the emission tensors. Unlike in case of the absorption tensors, here independent of n we always have two emission tensors. That is because at any step in the output-bit algorithm we output **L** if the numerator is less than the denominator and we output **R** if denominator is less than the numerator (cf. Definition 2.2). If we can not decide which of the numerator or denominator is bigger then we absorb more input. Thus we denote the two emission tensors by T_D (for the case of outputting **L**) and T_N (for the case of outputting **R**). Each time we output an **L** bit, we subtract the numerator from the denominator and each time we output an **R** we subtract the denominator from the numerator (cf. Definition 2.2). Thus the emission tensors become:

$$T_D=\begin{bmatrix} a_{2^n} & a_{2^n-1} & \cdots & a_1\\ b_{2^n}-a_{2^n} & b_{2^n-1}-a_{2^n-1} & \cdots & b_1-a_1\end{bmatrix},\quad T_N=\begin{bmatrix} a_{2^n}-b_{2^n} & \cdots & a_1-b_1\\ b_{2^n} & \cdots & b_1\end{bmatrix}$$

Next we generalise the functions f_1 and f_2.

Definition 4.13 Let $m_1, \cdots, m_{2^n} \in \mathbb{Z}$. We define their *sum of sign* to be $f_n(m_1, \cdots, m_{2^n}) = \operatorname{sgn} m_1 + \cdots + \operatorname{sgn} m_{2^n}$.

We use the sum of sign as a criteria to determine the sign of the multilinear polynomial whose coefficients are $m_1, \cdots, m_{2^n}$. This is shown for simple cases in the lemmata 4.2 and 4.7 and more generally in the following lemma:

Lemma 4.14

i) $f_n(a_1, \cdots, a_{2^n}) > 2^n - 2$ *then* $\forall x_1, \cdots, x_{2^n} > 0$ $L_n(a_1, \cdots, a_{2^n}) > 0$.

ii) $f_n(a_1, \cdots, a_{2^n}) < -2^n + 2$ *then* $\forall x_1, \cdots, x_{2^n} > 0$ $L_n(a_1, \cdots, a_{2^n}) < 0$.

iii) $f_1(f_n(a_1, \cdots, a_{2^n}) - (2^n - 2), f_n(b_1, \cdots, b_{2^n}) - (2^n - 2)) = 2$ *or*
$f_1(f_n(a_1, \cdots, a_{2^n}) + (2^n - 2), f_n(b_1, \cdots, b_{2^n}) + (2^n - 2)) = -2$ *then*
$\forall x_1, \cdots, x_{2^n} > 0$ $\mu_n(x_1, \cdots, x_{2^n}) > 0$.

iv) $f_1(f_n(a_1, \cdots, a_{2^n}) - (2^n - 2), f_n(b_1, \cdots, b_{2^n}) + (2^n - 2)) = 0$ *or*
$f_1(f_n(a_1, \cdots, a_{2^n}) + (2^n - 2), f_n(b_1, \cdots, b_{2^n}) - (2^n - 2)) = 0$ *then*
$\forall x_1, \cdots, x_{2^n} > 0$ $\mu_n(x_1, \cdots, x_{2^n}) < 0$.

Proof.

i) if $f_n(a_1, \cdots, a_{2^n}) > 2^n - 2$ then $f_n(a_1, \cdots, a_{2^n}) = 2^n - 1$ or $f_n(a_1, \cdots, a_{2^n}) = 2^n$. Thus we have at least $2^n - 1$ of $a_1, \cdots, a_{2^n}$ are greater than zero and none of them is less than 0. In either of the $2^n + 1$ cases the desired inequality holds.

ii) similar to i.

iii) Assume $f_1(f_n(a_1, \cdots, a_{2^n}) - (2^n - 2), f_n(b_1, \cdots, b_{2^n}) - (2^n - 2)) = 2$ then $f_n(f_n(a_1, \cdots, a_{2^n}) > (2^n - 2)$ and $f_n(b_1, \cdots, b_{2^n}) > (2^n - 2))$. If $x_1, \cdots, x_{2^n} > 0$ by i we have $L_n(a_1, \cdots, a_{2^n}) > 0$ and $L'_n(b_1, \cdots, b_{2^n}) > 0$ and hence $\mu_n(x_1, \cdots, x_{2^n}) > 0$. The case $f_1(f_n(a_1, \cdots, a_{2^n}) + (2^n - 2), f_n(b_1, \cdots, b_{2^n}) + (2^n - 2)) = -2$ is similar.

iv) similar to iii.

□

Next we present the general case of the sign algorithm. The input is the number of the variables n, the tensor T_n and the n Stern–Brocot binary sequences $\sigma_1, \cdots, \sigma_n \in \mathbf{SB}$. The output is an $n+2$-tuple $(s, T_{n_s}, \sigma_{1_s}, \cdots, \sigma_{n_s})$ consisting of the sign, the new coefficients tensor and the remaining parts of the n input sequences. We present the algorithm in the form of a recursive function $\mathcal{S}_n : T_{2\times 2^n}(\mathbb{Z}) \times \mathbf{SB}^n \longmapsto \{0, +1, -1\} \times T_{2\times 2^n}(\mathbb{Z}) \times \mathbf{SB}^n$. The recursion is on n as well as the input sequences. First of all when $n = 0$ then $\mathcal{S}_0$ is just determining the sign of the fraction $\frac{a}{b}$. In this case there is no input string so we output $((\operatorname{sgn} a)(\operatorname{sgn} b), \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right])$. For the case $n > 0$ we have to introduce some special tensors to deal with the case in which the input sequence σ_i $(1 \le i \le n)$ is the empty sequence.

Definition 4.15 We denote the $2\times 2^{n-1}$ tensor of the coefficients of $\mu_n(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n)$ by A_i. We assume that the outcome of $\mathcal{S}_{n-1}(A_i, \sigma_1, \cdots, \sigma_{i-1}, \sigma_{i+1}, \cdots, \sigma_n)$ is of the following form:

$$(s_i, B_{s_i}, \sigma_{s_1}, \cdots \sigma_{s_{i-1}}, \sigma_{s_{i+1}}, \cdots, \sigma_{s_n});$$

where B_{s_i} is the new $2\times 2^{n-1}$ tensor of coefficients. From B_{s_i} we construct the 2×2^n tensor $T_{n_{B_{s_i}}}$. The tensor $T_{n_{B_{s_i}}}$ is a tensor whose 2^{n-1} columns corresponding to the monomilas *not* including x_i are the columns from B_{s_i}, and the other 2^{n-1} columns are 0. As an example the tensor $T_{B_{s_1}}$ of Section 4.2 is obtained in this way.

When the input sequence σ_i $(1 \leq i \leq n)$ is the empty sequence, it means that one input sequence has vanished. Thus in order to calculate μ_n we can consider the coefficients of the monomials in which this vanishing variable occurs to be 0 [4]. This way, using the above notation, we output (as part of the output $n+2$-tuple) the tensor $T_{n_{B_{s_i}}}$. We mention some useful properties of the matrices A_i and $T_{B_{s_i}}$:

Lemma 4.16 *Let $\sigma_i = []$ and A_i, B_{s_i} and $T_{B_{s_i}}$ be defined as in 4.15. Then*

i) $\mu_{n-1_{A_i}}(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n) = \mu_{n_{T_n}}(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n)$.

ii) $\mu_{n-1_{B_{s_i}}}(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n) = \mu_{n_{T_{B_{s_i}}}}(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n)$.

Proof. Immediate by induction on n and unfolding the definitions. □

In the following algorithm we use the shorthand notation $\kappa_1 = f_n(a_1, \cdots, a_{2^n})$ and $\kappa_2 = f_n(b_1, \cdots, b_{2^n})$.

Sign algorithm for n variables.

$$
\begin{cases}
\mathbb{S}_n(T_n, \sigma_1, \cdots, \sigma_{i-1}, [], \sigma_{i+1}, \cdots, \sigma_n) := (s_i, T_{n_{B_{s_i}}}, \sigma_{s_1}, \cdots \sigma_{s_{i-1}}, [], \sigma_{s_{i+1}}, \cdots, \sigma_{s_n}) \\
\mathbb{S}_n(T_n, \sigma_1, \cdots, \sigma_n) := \\
\quad \begin{cases}
\textbf{if } \forall i < 2^n \ a_i = 0 & \begin{cases}
((\operatorname{sgn} a_{2^n})(\operatorname{sgn} b_{2^n}), T_n, \sigma_1, \cdots, \sigma_n) & \textbf{if } \forall i < 2^n \ b_i = 0, \\
(\operatorname{sgn} a_{2^n}, T_n, \sigma_1, \cdots, \sigma_n) & \textbf{else if } \kappa_2 > 2^n - 2, \\
(-\operatorname{sgn} a_{2^n}, T_n, \sigma_1, \cdots, \sigma_n) & \textbf{else if } \kappa_2 < -2^n + 2, \\
\mathcal{X}_{n\,T_n}(\sigma_1, \cdots, \sigma_n) & \textbf{otherwise}.
\end{cases} \\
\textbf{otherwise} & \begin{cases}
\textbf{if } \forall i < 2^n \ b_i = 0 & \begin{cases}
(\operatorname{sgn} b_{2^n}, T_n, \sigma_1, \cdots, \sigma_n) & \textbf{else if } \kappa_1 > 2^n - 2, \\
(-\operatorname{sgn} b_{2^n}, T_n, \sigma_1, \cdots, \sigma_n) & \textbf{else if } \kappa_1 < -2^n + 2, \\
\mathcal{X}_{n\,T_n}(\sigma_1, \cdots, \sigma_n) & \textbf{otherwise}.
\end{cases} \\
\textbf{otherwise} & \begin{cases}
(+1, T_n, \sigma_1, \cdots, \sigma_n) & \textbf{if } f_1(\kappa_1 - (2^n - 2), \kappa_2 - (2^n - 2)) = 2 \\
 & \vee f_1(\kappa_1 + (2^n - 2), \kappa_2 + (2^n - 2)) = -2, \\
(-1, T_n, \sigma_1, \cdots, \sigma_n) & \textbf{else if } f_1(\kappa_1 - (2^n - 2), \kappa_2 + (2^n - 2)) = 0 \\
 & \vee f_1(\kappa_1 + (2^n - 2), \kappa_2 - (2^n - 2)) = 0, \\
\mathcal{X}_{n\,T_n}(\sigma_1, \cdots, \sigma_n) & \textbf{otherwise}.
\end{cases}
\end{cases}
\end{cases}
$$

The function $\mathcal{X}_n : T_{2\times 2^n}(\mathbb{Z}) \times \mathbf{SB}^n \longmapsto \{0, +1, -1\} \times T_{2\times 2^n}(\mathbb{Z}) \times \mathbf{SB}^n$ is the absorption step and is defined by pattern matching on the head elements of the input sequences using the T_{π_i} tensors defined earlier in this section. If the head elements of the n sequnces are the the sequence π_i then:

$$\mathcal{X}_n(T_n, \pi_i(1)\sigma'_1, \cdots, \pi_i(n)\sigma'_n) := \mathbb{S}_n(T_{\pi_i}, \sigma'_1, \cdots, \sigma'_n)$$

Next we turn our attention to the generalised form of the output-bit algorithm. First we generalise the Definitions 4.3 and 4.8:

Definition 4.17 Let $p_1, \cdots, p_{2^n}, q_1, \cdots, q_{2^n} \in \mathbb{Z}$. We define the predicate:

$$\psi_n(p_1, \cdots, p_{2^n}; q_1, \cdots, q_{2^n}) := \bigvee_{1 \leq i \leq 2^n} \left(\left(\bigwedge_{k \neq i} q_k \leq p_k \right) \wedge q_i < p_i \right)$$

[4] That is to say, we consider $\mu_n(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n)$ as another multilinear fractional form $\mu'_n(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n)$.

Again we state those properties of these predicate that we use in the algorithm:

Lemma 4.18

i) If $\psi_n(a_1,\cdots,a_{2^n};b_1,\cdots,b_{2^n})$ *then* $\forall x_1,\cdots,x_{2^n}>0\ L_n(x_1,\cdots,x_{2^n})>L'_n(x_1,\cdots,x_{2^n})$.

ii) If $\neg\psi_n(a_1,\cdots,a_{2^n};b_1,\cdots,b_{2^n})\wedge\neg\psi_n(b_1,\cdots,b_{2^n};a_1,\cdots,a_{2^n})$ *then*

$$\exists x_1,\cdots,x_{2^n}{>}0.\ L_n(x_1,\cdots,x_{2^n})=L'_n(x_1,\cdots,x_{2^n})$$

In the generalised output-bit algorithm, the input is the number of the variables n, the tensor T_n and the n Stern–Brocot binary sequences $\sigma_1,\cdots,\sigma_n\in\mathbf{SB}$. We present the algorithm as a recursive function $\mathcal{B}_n : T_{2\times 2^n}(\mathbb{Z})\times\mathbf{SB}^n\longmapsto\mathbf{SB}$. The recursion is on n as well as the input sequences and the tensor of coefficients [5]. In case one of the input sequences is the empty sequences, we need to call the $\mathcal{B}_{n-1}$ function and perform the recursion on n. Since in the base cases $n=1$ and $n=2$ we had the precondition that for empty input sequences $L_1(1),L'_1(1)>0$ and $L_2(1,1),L'_2(1,1)>0$; this precondition will mount to the case of arbitrary n. Thus if for some $1\leq i\leq n$ the input sequence σ_i is the empty sequence we need to require that $L_n(1,1,\cdots,1)=\sum_{i=1}^{2^n}a_i>0$ and $L'_n(1,1,\cdots,1)=\sum_{i=1}^{2^n}b_i>0$. Moreover note that in the case when one of the input sequences is empty we have a recursive step on n with a different coefficients matrix. Hence we need the following lemma which states that the preconditions are preserved during the recursive step:

Lemma 4.19 *For a tensor* $T\in T_{2\times 2^n}(\mathbb{Z})$ *by* $row_1(T)$ *we mean the sum of the elements in the first row of the tensor and by* $row_2(T)$ *we mean the sum of the elements in the second row of the tensor. Suppose* $row_1(T_n),row_2(T_n)>0$. *Then* $row_k(A_i)>0 (k=1,2)$.

Proof. A_i is obtained from T_n by simplifying $\mu_{n_{T_n}}(x_1,\cdots,x_{i-1},1,x_{i+1},\cdots,x_n)$ and writing it as $\mu_{n-1_{A_i}}(x_1,\cdots,x_{i-1},x_{i+1},\cdots,x_n)$. By induction on n one can easily see that during the simplification, $row_k(T_n)$ remains equal to $row_k(A_i)$. □

We are ready to present the output-bit algorithm for n variables.
output-bit algorithm for $a_i,b_i\in\mathbb{Z},\sigma_i,\in\boldsymbol{SB}, row_1(T_n),row_2(T_n)>0$.

$$\begin{cases}\mathcal{B}_n(T_n,\sigma_1,\cdots,\sigma_{i-1},[],\sigma_{i+1},\cdots,\sigma_n):=\mathcal{B}_{n-1}(A_i,\sigma_1,\cdots,\sigma_{i-1},\sigma_{i+1},\cdots,\sigma_n)\\ \mathcal{B}_n(T_n,\sigma_1,\cdots,\sigma_n):=\begin{cases}\mathbf{R}\ (\mathcal{B}_n(T_N,\sigma_1,\cdots,\sigma_n)) & \textbf{if } \psi_n(a_1,\cdots,a_{2^n};b_1,\cdots,b_{2^n}),\\ \mathbf{L}\ (\mathcal{B}_n(T_D,\sigma_1,\cdots,\sigma_n)) & \textbf{else if } \psi_n(b_1,\cdots,b_{2^n};a_1,\cdots,a_{2^n}),\\ \mathcal{X}'_n(T_n,\sigma_1,\cdots,\sigma_n) & \textbf{otherwise}.\end{cases}\end{cases}$$

Where the function $\mathcal{X}'_n : T_{2\times 2^n}(\mathbb{Z})\times\mathbf{SB}^n\longmapsto\mathbf{SB}$ is defined by pattern matching on the head element of input similar to the above $\mathcal{X}_n$ function; an in the other absorption cases we use the tensors T_{π_i} defined earlier in this section:

$$\mathcal{X}'_n(T_n,\pi_i(1)\sigma'_1,\cdots,\pi_i(n)\sigma'_n):=\mathcal{B}_n(T_{\pi_i},\sigma'_1,\cdots,\sigma'_n)$$

Finally combining the above two algorithms we present the algorithm to compute the multilinear fractional form μ_n. Given n the number of variables, the tensors T_n of the coefficients and the Stern–Brocot binary sequences, this algorithm will calculate $\mu_{n_{T_n}}([\![\sigma_1]\!],\cdots,[\![\sigma_n]\!])$. The main steps are taken in the output-bit algorithm. We only need to make sure that when we call the output-bit algorithm the preconditions hold. in order to achieve this we use the outcome of the sign algorithm to correct the signs of the coefficients to satisfy the precondition. Thus assuming

[5] For this to work we need to define a well-ordering on the set of tensors. This is always possible (See the discussion before Theorem 4.21 and [14])

that $\mathcal{S}_n(T_n, \sigma_1, \cdots, \sigma_n) = (s, T_{n_s}, \sigma_{1_s}, \cdots, \sigma_{n_s})$, and that $T_{n_s} = \begin{bmatrix} a_{s_{2^n}} & a_{s_{2^n-1}} & \cdots & a_{s_0} \\ b_{s_{2^n}} & b_{s_{2^n-1}} & \cdots & b_{s_0} \end{bmatrix}$ we use the notations $T^-_{n_s}$,$T^-_{n_{s_1}}$ and $T^-_{n_{s_2}}$ to denote the modified versions of the T_{n_s} tensor. In $T^-_{n_s}$ we negate all the elements of the tensor T_{n_s}. In $T^-_{n_{s_1}}$ (resp. $T^-_{n_{s_2}}$) we negate only the elements in the second row (resp. first row) of T_{n_s} (cf. T^-_s,$T^-_{s_1}$ and $T^-_{s_2}$ in Section 4.2). Next we present the generalised form of the Definition 4.20:

Definition 4.20 For $T_n = \begin{bmatrix} a_{2^n} & a_{2^n-1} & \cdots & a_0 \\ b_{2^n} & b_{2^n-1} & \cdots & b_0 \end{bmatrix}$ we define the predicate *same_ratio* (T_n) as:

$$\textit{same\_ratio}\ (T_n) := \bigwedge_{1 \leq i,j \leq n} a_i b_j = a_j b_i.$$

We present the algorithm in the form of a function $\mathcal{M}_n : T_{2\times(2\times 2)}(\mathbb{Z}) \times \mathbf{SB}^n \longmapsto \mathbf{SSB}$. Note that there is no recursive calls in this function:

multilinear algorithm for n variables.

$$\begin{cases} \mathcal{M}_n(T_n, \sigma_1, \cdots, \sigma_n) := & \\ \quad \begin{cases} \textbf{if } \textit{same\_ratio}\ (T_n) & \begin{cases} \textbf{ZERO} & \textbf{if } s = 0, \\ \textbf{if } s = 1 & \begin{cases} \textbf{POS } \mathcal{B}_n(T_{n_s}, \sigma_{1_s}, \cdots, \sigma_{n_s}) & \textbf{if } \sum\limits_{1 \leq i \leq 2^n} a_{s_i} > 0, \\ \textbf{POS } \mathcal{B}_n(T^-_{n_s}, \sigma_{1_s}, \cdots, \sigma_{n_s}) & \textbf{otherwise}. \end{cases} \\ \textbf{if } s = -1 & \begin{cases} \textbf{NEG } \mathcal{B}_n(T^-_{n_{s_1}}, \sigma_{1_s}, \cdots, \sigma_{n_s}) & \textbf{if } \sum\limits_{1 \leq i \leq 2^n} a_{s_i} > 0, \\ \textbf{NEG } \mathcal{B}_n(T^-_{n_{s_2}}, \sigma_{1_s}, \cdots, \sigma_{n_s}) & \textbf{otherwise}. \end{cases} \end{cases} \\ \textbf{otherwise} & \ulcorner \frac{a_i}{b_i} \urcorner \quad \textbf{for the first } 1 \leq i \leq 2^n \textbf{ s.t. } b_i \neq 0. \end{cases} \end{cases}$$

After defining the generalised form of the algorithms we proceed to proof the correctness of these family of the algorithms. Since our algorithms are given as recursive function, we need to follow *well-founded induction* on recursive calls to prove properties about these functions. For well-founded induction one needs to have a well-ordering on the structure which is decreasing in succesive recursive calls. In our algorithms we have recursions on three different structres:

(1) Recursion on natural numbers. This leads to ordinary induction on natural numbers.

(2) Recursion on the structure of binary sequences. This leads to a well-ordering on the set of finite binary sequences, which is basically the (partial) oredring of sequences according to their length. According to this well-ordering, the new input sequnces (consider as an n-tuple) after an absorption step should be considered less than the n-tuple of input sequnces before the absorption. Therefore we use this induction scheme to prove th properties that are invariant during the absorption steps.

(3) Recursion on the structure of tensor of the coefficients. This leads to a well-ordering on the set of 2×2^n tensors over positive integers, which is the ordering of tensors according to sum of their elements. According to this well-ordering, the tensor of coefficients after an emission step should be considered less than the tensor of the coefficients before the emission. Hence we use this inuduction scheme weherever our algorithm contains an emission step (e.g. the function $\mathcal{B}_n$).

Sometimes we need a combination of the above orderings. In that case we form a lexicographic order in which the component oredrs are the above well-orderings. One can prove that this lexicographic order is also well-founded, and hence one can use well-founded induction on the lexicographic order. In the proof of the following theorem we use well-founded induction on the above structures.

Theorem 4.21 (Correctness of Multiliear Algorithm) *Let* $\mu_{n_{T_n}}(x) = \frac{L_n(x_1,\cdots,x_n)}{L'_n(x_1,\cdots,x_n)}$ *where* $T_n = \begin{bmatrix} a_{2^n} & a_{2^n-1} & \cdots & a_0 \\ b_{2^n} & b_{2^n-1} & \cdots & b_0 \end{bmatrix} \in T_{2\times 2^n}(\mathbb{Z})$. *Let* $\sigma_1, \cdots, \sigma_n \in \boldsymbol{SB}$.

i) Assume $\mathcal{S}_n(T_n, \sigma_1, \cdots, \sigma_n) = (s, T_{n_s}, \sigma_{1_s}, \cdots, \sigma_{n_s})$. *Then* $s = \operatorname{sgn} \mu_{n_{T_n}}([\![\sigma_1]\!], \cdots, [\![\sigma_n]\!])$.

ii) If $\sum_{1\le i\le 2^n} a_i > 0$ *and* $\sum_{1\le i\le 2^n} b_i > 0$ *then* $\ulcorner \mu_{n_{T_n}}([\![\sigma_1]\!], \cdots, [\![\sigma_n]\!])\urcorner = \mathcal{B}_n(T_n, \sigma_1, \cdots, \sigma_n)$.

iii) $\ulcorner \mu_{n_{T_n}}([\![\sigma_1]\!], \cdots, [\![\sigma_n]\!])\urcorner = \mathcal{M}_n(T_n, \sigma_1, \cdots, \sigma_n)$.

Proof.

Part i) We prove by induction on n. The case $n = 0$ trivially holds as sign algorithm is degenerate in this case. Consider $\mathcal{S}_{n+1}(T_{n+1}, \sigma_1, \cdots, \sigma_{n+1}) = (s, T_{n+1_s}, \sigma_{1_s}, \cdots, \sigma_{n+1_s})$. First we consider the case when for some i we have $\sigma_i = []$. According to the definition of $\mathcal{S}_{n+1}$ we have $(s, T_{n+1_s}, \sigma_{1_s}, \cdots, \sigma_{n+1_s}) = (s_i, T_{n_{B_{s_i}}}, \sigma_{s_1}, \cdots \sigma_{s_{i-1}}, [], \sigma_{s_{i+1}}, \cdots, \sigma_{s_{n+1}})$ and therefore

$$s = s_i \tag{4.21.1}$$

Recall that:

$$\mathcal{S}_n(A_i, \sigma_1, \cdots, \sigma_{i-1}, \sigma_{i+1}, \cdots, \sigma_n) = (s_i, B_{s_i}, \sigma_{s_1}, \cdots \sigma_{s_{i-1}}, \sigma_{s_{i+1}}, \cdots, \sigma_{s_n}).$$

By induction hypothesis:

$$s_i = \operatorname{sgn} \mu_{n_{A_i}}([\![\sigma_1]\!], \cdots, [\![\sigma_{i-1}]\!], [\![\sigma_{i+1}]\!], \cdots, [\![\sigma_{n+1}]\!]) \tag{4.21.2}$$

But according to the Lemma 4.16.i we have:

$$\mu_{n_{A_i}}(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_{n+1}) = \mu_{n+1_{T_{n+1}}}(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_{n+1}) \tag{4.21.3}$$

Therefore by (4.21.1), (4.21.2) and (4.21.3) the result follows.
Now assume that neither of the input sequences are empty. In this case in addition to our original induction on n we proceed by well-founded induction on the structure of the binary input sequences. That is to say we assume the conclusion to be true for for all $\tau_1, \cdots, \tau_n$ where $\mathsf{length}(\tau_i) \le \mathsf{length}(\sigma_i)$ and at least for some j $\tau_j \neq \sigma_j$. Then we follow the definition of $\mathcal{S}_{n+1}$ and consider different cases. First assume:

$$a_i = 0 \quad (\text{for every } i < 2^{n+1}). \tag{4.21.4}$$

Chasing through the definition of $\mathcal{S}_{n+1}$ we observe that in this case we have four possibilities:

1. $\forall i < 2^{n+1} b_i = 0$: Here $s = (\operatorname{sgn} a_{2^{n+1}})(\operatorname{sgn} b_{2^{n+1}})$. Since all the other coefficients are zero and all $x_i > 0$, after simplification we get

   $$\mu_{n+1}(x_1, \cdots, x_{n+1}) = \frac{a_{2^{n+1}}}{b_{2^{n+1}}}.$$

   Thus in this case we obtain

   $$\operatorname{sgn} \mu_{n+1_{T_{n+1}}}(x_1, \cdots, x_{n+1}) = (\operatorname{sgn} a_{2^{n+1}})(\operatorname{sgn} b_{2^{n+1}}) = s.$$

2. $\kappa_2 > 2^{n+1} - 2$: Here $s = \operatorname{sgn} a_{2^{n+1}}$. By Lemma 4.14. i we have $L'_{n+1}(b_1, \cdots, b_{2^{n+1}}) > 0$. Note that by (4.21.4) the sign of numerator is the sign of $a_{2^{n+1}}$. Therefore we get

   $$\operatorname{sgn} \mu_{n+1_{T_{n+1}}}(x_1, \cdots, x_{n+1}) = \operatorname{sgn} a_{2^{n+1}} = s.$$

3. $\kappa_2 < -2^{n+1}+2$: Here $s = -\text{sgn}\, a_{2^{n+1}} = 0$. By Lemma 4.14. ii we have $L'_{n+1}(b_1, \cdots, b_{2^{n+1}}) < 0$. Note that by (4.21.4) the sign of numerator is the sign of $a_{2^{n+1}}$. Therefore we get

$$\text{sgn}\, \mu_{n+1_{T_{n+1}}}(x_1, \cdots, x_{n+1}) = -\text{sgn}\, a_{2^{n+1}} = s.$$

4. In the case where neither of the above three cases hold we have to use the hypothesis of the well-founded induction. Here $\mathcal{S}_{n+1}(T_{\pi_i}, \sigma'_1, \cdots, \sigma'_{n+1}) = (s, T_{n+1_s}, \sigma_{1_s}, \cdots, \sigma_{n+1_s})$. By well-founded induction's hypothesis we know that since $\sigma'_1, \cdots, \sigma'_{n+1}$ are structurally smaller than $\sigma_1, \cdots, \sigma_{n+1}$ therefore

$$s = \text{sgn}\, \mu_{n+1_{T_{\pi_i}}}([\![\sigma'_1]\!], \cdots, [\![\sigma'_{n+1}]\!]). \tag{4.21.5}$$

But according to the equation (4.12.1) on page 13 we have:

$$\begin{aligned}\mu_{n+1_{T_{n+1}}}([\![\sigma_1]\!], \cdots, [\![\sigma_{n+1}]\!]) &= \mu_{n+1_{T_{n+1}}}(\phi(\pi_i(1), [\![\sigma'_1]\!]), \cdots, \phi(\pi_i(n+1), [\![\sigma'_{n+1}]\!])) \\ &= \mu_{n+1_{T_{\pi_i}}}([\![\sigma'_1]\!], \cdots, [\![\sigma'_{n+1}]\!]).\end{aligned} \tag{4.21.6}$$

The result follows from (4.21.5) and (4.21.6).

Next assume:

$$(\neg \forall i < 2^{n+1} a_i = 0) \wedge \forall i < 2^{n+1} b_i = 0. \tag{4.21.7}$$

According to the algorithm for $\mathcal{S}_{n+1}$, here we have three different cases which are similar to the last three of the above four cases and therefore we omit the proof.

The last case we consider is the case in which

$$(\neg \forall i < 2^{n+1} a_i = 0) \wedge (\neg \forall i < 2^{n+1} b_i = 0). \tag{4.21.8}$$

Here we have to consider three different cases:

(a) Assume $f_1(\kappa_1 - (2^n - 2), \kappa_2 - (2^n - 2)) = 2$ or $f_1(\kappa_1 + (2^n - 2), \kappa_2 + (2^n - 2)) = -2$. Here $s = +1$ and by Lemma 4.14. iii we immediately get:

$$\text{sgn}\, \mu_{n+1_{T_{n+1}}}(x_1, \cdots, x_{n+1}) = +1 = s.$$

(b) $f_1(\kappa_1 - (2^n - 2), \kappa_2 + (2^n - 2)) = 0$ or $f_1(\kappa_1 + (2^n - 2), \kappa_2 - (2^n - 2)) = 0$. This is similar to the case (a) and follows immediately from Lemma 4.14. iv.

(c) In this case neither of the above conditions holds. Here we use the same reasoning as in the case 4. above and use the well-founded induction.

Part ii) We start by induction on n. For the case where $n = 0$ the conclusion is trivial because $T_0 = \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$ and $\mathcal{B}_0$ is the the $\ulcorner \_ \urcorner$ function of Section 2. Therefore both side of the conclusion become $\ulcorner \frac{a}{b} \urcorner$. For the case of $n+1$ we first consider the case when for some $1 \leq i \leq n+1$ we have $\sigma_i = []$. According to the definition we have :

$$\mathcal{B}_{n+1}(T_{n+1}, \sigma_1, \cdots, \sigma_{i-1}, [], \sigma_{i+1}, \cdots, \sigma_{n+1}) = \mathcal{B}_n(A_i, \sigma_1, \cdots, \sigma_{i-1}, \sigma_{i+1}, \cdots, \sigma_{n+1}). \tag{4.21.9}$$

By Lemma 4.19 we know that $row_1(A_i), row_2(A_i) > 0$. Therefore we can use the induction hypothesis for the right hand side. Then we obtain:

$$\mathcal{B}_n(A_i, \sigma_1, \cdots, \sigma_{i-1}, \sigma_{i+1}, \cdots, \sigma_{n+1}) = \ulcorner \mu_{n_{A_i}}([\![\sigma_1]\!], \cdots, [\![\sigma_{i-1}]\!], [\![\sigma_{i+1}]\!], \cdots, [\![\sigma_{n+1}]\!]) \urcorner. \tag{4.21.10}$$

But according to Lemma 4.16.i we have:

$$\mu_{n_{A_i}}(x_1,\cdots,x_{i-1},x_{i+1},\cdots,x_{n+1}) = \mu_{n+1_{T_{n+1}}}(x_1,\cdots,x_{i-1},1,x_{i+1},\cdots,x_{n+1}) \tag{4.21.11}$$

And from (4.21.9), (4.21.10) and (4.21.11) the conclusion follows.
Next we assume that neither of the input sequences are empty. For the proof of this part we apply the well-founded induction on the structure of the binary input sequences and the tensors of coefficients.
Following the definition of $\mathcal{B}_{n+1}$ we have to consider three cases:

1. $\psi_n(a_1,\cdots,a_{2^n};b_1,\cdots,b_{2^n})$. Here we have

$$\mathcal{B}_{n+1}(T_{n+1},\sigma_1,\cdots,\sigma_{n+1}) = \mathbf{R}\ (\mathcal{B}_{n+1}(T_N,\sigma_1,\cdots,\sigma_{n+1})). \tag{4.21.12}$$

   By the (well-founded) induction hypothesis we have:

$$\mathcal{B}_{n+1}(T_N,\sigma_1,\cdots,\sigma_{n+1}) = \ulcorner \mu_{n+1_{T_N}}([\![\sigma_1]\!],\cdots,[\![\sigma_{n+1}]\!])\urcorner. \tag{4.21.13}$$

   By the definition of T_N we have:

$$\mu_{n+1_{T_N}}(x_1,[\![\sigma_{n+1}]\!]) = \frac{L_{n+1}(x_1,\cdots,x_n) - L'_{n+1}(x_1,\cdots,x_n)}{L'_{n+1}(x_1,\cdots,x_n)}. \tag{4.21.14}$$

   Note that by Lemma 4.18. i we have

$$L_{n+1}(x_1,\cdots,x_n) > L'_{n+1}(x_1,\cdots,x_n).$$

   Whence using Definition 2.2 and the equations (4.21.13) and (4.21.14) we derive:

$$\begin{aligned}\mathbf{R}\ (\mathcal{B}_{n+1}(T_N,\sigma_1,\cdots,\sigma_{n+1})) &= \mathbf{R}\ \ulcorner \frac{L_{n+1}(x_1,\cdots,x_{n+1}) - L'_{n+1}(x_1,\cdots,x_{n+1})}{L'_{n+1}(x_1,\cdots,x_{n+1})}\urcorner \\ &= \ulcorner \frac{L_{n+1}(x_1,\cdots,x_{n+1})}{L'_{n+1}(x_1,\cdots,x_{n+1})}\urcorner \\ &= \ulcorner \mu_{n+1_{T_{n+1}}}(x_1,\cdots,x_{n+1})\urcorner,\end{aligned} \tag{4.21.15}$$

   and from here by (4.21.12) the desired result follows.
2. $\psi_n(b_1,\cdots,b_{2^n};a_1,\cdots,a_{2^n})$. This case is similar to the previous case.
3. In the case where none of the above two conditions holds we have

$$\mathcal{B}_{n+1}(T_{n+1},\sigma_1,\cdots,\sigma_{n+1}) = \mathcal{B}_{n+1}(T_{\pi_i},\sigma'_1,\cdots,\sigma'_{n+1}). \tag{4.21.16}$$

   Since $\sigma'_1,\cdots,\sigma'_{n+1}$ are structurally smaller than $\sigma_1,\cdots,\sigma_{n+1}$ therefore by (well-founded) induction hypothesis we know that:

$$\mathcal{B}_{n+1}(T_{\pi_i},\sigma'_1,\cdots,\sigma'_{n+1}) = \ulcorner \mu_{n+1_{T_{\pi_i}}}([\![\sigma'_1]\!],\cdots,[\![\sigma'_{n+1}]\!])\urcorner. \tag{4.21.17}$$

   But according to the equation (4.12.1) on page 13 we have:

$$\begin{aligned}\mu_{n+1_{T_{\pi_i}}}([\![\sigma'_1]\!],\cdots,[\![\sigma'_{n+1}]\!]) &= \mu_{n+1_{T_{n+1}}}(\phi(\pi_i(1),[\![\sigma'_1]\!]),\cdots,\phi(\pi_i(n+1),[\![\sigma'_{n+1}]\!])) \\ &= \mu_{n+1_{T_{n+1}}}([\![\sigma_1]\!],\cdots,[\![\sigma_{n+1}]\!]);\end{aligned} \tag{4.21.18}$$

   The final result follows from (4.21.16), (4.21.17) and (4.21.18).

Part iii) Assuming $\mathcal{S}_n(T_n, \sigma_1, \cdots, \sigma_n) = (s, T_{n_s}, \sigma_{1_s}, \cdots, \sigma_{n_s})$ we consider different cases by case distinction on s. Case $s = 0$ is trivial. We consider the case $s = +1$. Assume $row(T_{n_s}) > 0$. In this case unfolding the definition and using part ii we can write:

$$\begin{aligned}\mathcal{M}_n(T_n, \sigma_1, \cdots, \sigma_n) &= \mathbf{POS}\, \mathcal{B}_n(T_{n_s}, \sigma_{1_s}, \cdots, \sigma_{n_s}) \\ &= \mathbf{POS}\, \ulcorner \mu_{n_{T_{n_s}}}([\![\sigma_{1_s}]\!], \cdots, [\![\sigma_{n_s}]\!]) \urcorner \\ &= \ulcorner \mu_{n_{T_{n_s}}}([\![\sigma_{1_s}]\!], \cdots, [\![\sigma_{n_s}]\!]) \urcorner\end{aligned}$$

Therefore we only need to prove:

$$\mu_{n_{T_{n_s}}}([\![\sigma_{1_s}]\!], \cdots, [\![\sigma_{n_s}]\!]) = \mu_{n_{T_n}}([\![\sigma_1]\!], \cdots, [\![\sigma_n]\!]) \tag{4.21.19}$$

We prove this by induction on n. The case $n = 0$ is trivial. In order to prove the general case we need to examine the way $(T_{n+1_s}, \sigma_{1_s}, \cdots, \sigma_{n+1_s})$ is generated. By investigating the $\mathcal{S}_{n+1}$ one observes that the only steps in the algorithm when we change the tensor of the coefficients, is when T_{n+1} becomes $T_{n+1_{B_{s_i}}}$ in the output or T_{n+1} is modified to T_{π_i} in the recursive step. We consider each of these two cases separately:

1. In the case when T_{n+1} transforms to $T_{n+1_{B_{s_i}}}$ we necessarily have $\sigma_i = []$. Assume

   $$\begin{aligned}&\mathcal{S}_n(A_i, \sigma_1, \cdots, \sigma_{i-1}, \sigma_{i+1}, \cdots, \sigma_{n+1}) = (s_i, B_{s_i}, \sigma_{s_1}, \cdots \sigma_{s_{i-1}}, \sigma_{s_{i+1}}, \cdots, \sigma_{s_{n+1}}), \\ &\mathcal{S}_{n+1}(T_{n+1}, \sigma_1, \cdots, \sigma_{i-1}, [], \sigma_{i+1}, \cdots, \sigma_{n+1}) = \\ &\qquad\qquad (s_i, T_{n+1_{B_{s_i}}}, \sigma_{s_1}, \cdots, \sigma_{s_{i-1}}, [], \sigma_{s_{i+1}}, \cdots, \sigma_{s_{n+1}}).\end{aligned}$$

   Since A_i is a 2×2^n tensor, by induction hypothesis we have:

   $$\begin{aligned}&\mu_{n_{A_i}}([\![\sigma_1]\!], \cdots, [\![\sigma_{i-1}]\!], [\![\sigma_{i+1}]\!], \cdots, [\![\sigma_{n+1}]\!]) = \\ &\qquad\qquad \mu_{n_{B_{s_i}}}([\![\sigma_{1_s}]\!], \cdots, [\![\sigma_{s_{i-1}}]\!], [\![\sigma_{s_{i+1}}]\!], \cdots, [\![\sigma_{n_s}]\!])\end{aligned} \tag{4.21.20}$$

   By Lemma 4.16.i we have

   $$\begin{aligned}&\mu_{n_{A_i}}([\![\sigma_1]\!], \cdots, [\![\sigma_{i-1}]\!], [\![\sigma_{i+1}]\!], \cdots, [\![\sigma_{n+1}]\!]) = \\ &\qquad\qquad \mu_{n+1_{T_{n+1}}}([\![\sigma_1]\!], \cdots, , [\![\sigma_{i-1}]\!], 1, [\![\sigma_{i+1}]\!], \cdots, [\![\sigma_n]\!])\end{aligned} \tag{4.21.21}$$

   And by Lemma 4.16.ii we have

   $$\begin{aligned}&\mu_{n_{B_{s_i}}}([\![\sigma_{1_s}]\!], \cdots, [\![\sigma_{s_{i-1}}]\!], [\![\sigma_{s_{i+1}}]\!], \cdots, [\![\sigma_{n_s}]\!]) = \\ &\qquad\qquad \mu_{n+1_{T_{n+1_{B_{s_i}}}}}([\![\sigma_1]\!], \cdots, , [\![\sigma_{i-1}]\!], 1, [\![\sigma_{i+1}]\!], \cdots, [\![\sigma_n]\!])\end{aligned} \tag{4.21.22}$$

   And the proof of (4.21.19) is obtained from (4.21.20), (4.21.21) and (4.21.22).
2. In the case when T_{n+1} is modified to T_{π_i}. As in the other absorption steps we argue by well-founded induction on the structure of the $n+1$-tuple of input sequnces and we note that by equation (4.12.1) on page 13 we have

   $$\mu_{n+1_{T_{\pi_i}}}([\![\sigma'_1]\!], \cdots, [\![\sigma'_{n+1}]\!]) = \mu_{n+1_{T_{n+1}}}([\![\sigma_1]\!], \cdots, [\![\sigma_{n+1}]\!]).$$

   Hence (4.21.19) holds in this case.

This completes the proof of (4.21.19) by induction. Therefore in the case when $s = +1$ and $row(T_{n_s}) > 0$ we have proven the conclusion. We omit the proof for the remainig three cases ($row(T_{n_s}) \leq 0$ or $s = -1$) as it is identical to the above argument.

□

Hence we have shown the correctness of our algorithms for exact rational arithmetic based on Stern–Brocot representation. For the case $n = 2$ (the quaratic algorithms) this proof has been formalised in theorem prover Coq (See Section 6 and [14]).

5 Exact Real Arithmetic

In this section we discuss a possible representation for real numbers based on Stern–Brocot tree. The proofs are omitted and will be presented in an upcoming work of the author. In Section 2 we showed that one can encode the set of rational numbers as the set of finite binary sequences. Allowing the sequences to be infinite streams we get a representation for the set of real numbers. To be more precise one can prove the following lemma using the approximations given in each level of Stern–Brocot tree:

Lemma 5.1 *If α is an infinite stream of* $\mathbf{L}$ *and* $\mathbf{R}$ *and $\alpha \neq \mathbf{R}^\infty$, then decoding of its initial segments using the function defined in 2.4, forms a Cauchy sequence of rational numbers.*

After that, one can extend the set of finite sequences $\mathbf{SB}$ to contain all infinite binary streams. We call this new set $\mathbf{SB}^\omega$. We can extend the function $[\![\_]\!] : \mathbf{SB} \longrightarrow \mathbb{Q}^+$ to a function from all infinite streams:

$$[\![\alpha]\!] := \lim_{n\to\infty} [\![\alpha|_n]\!].$$

Then one might attempt to use the algorithms given in Section 4 to compute arithmetic operations. But the set $\mathbf{SB}^\omega$ does not have enough redundancy to be useful for computations. For example if we want to add $1-\sqrt{2}$ and $\sqrt{2}$ using the quadratic algorithm of Section 4.2, after absorbing any initial segment of the input streams we are still unable to output an element of the output because we do not know whether the result is bigger or smaller than 1. This is because in general it is undecidable whether a real number is a rational number. A similar problem arises with the usual decimal expansion of real numbers while multiplication by 3 is not a total computable function. The way around this is extend our alphabet in order to get what is known as an *admissible* representation. Admissible representations are those which arise out of the Cantor space topology on the set of streams [20].

There are various ways of forming a ternary representation based on Stern–Brocot representation. We add one new symbol to our alphabet and consider ternary streams instead of binary streams. Thus we might consider the set of streams over the 3-element alphabet $\{\mathbf{L}, \mathbf{R}, \mathbf{M}\}$.

To interpret the effect of the additional letter we introduce some definitions from [8]. Recall that a *Möbius map* is a map

$$f : x \longmapsto \frac{ax+b}{cx+d}$$

with $a, b, c, d \in \mathbb{R}$. We denote a Möbius map by its coefficient matrix $\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$. Let $\mathbb{R}^\star$ be the compactification of interval $(0, +\infty)$ by adding 0 and $+\infty$. A *d-contraction* is a map $f : \mathbb{R}^\star \longrightarrow \mathbb{R}^\star$ which is twice differentiable, has a unique fixpoint and for each $x \in \mathbb{R}^\star$ we have $0 < f'(x)$. Let Σ be a finite set of alphabet. Let $\phi_\_$ be a map that assigns to each element α of Σ a d-contraction ϕ_α. We extend this in the following way to a map that assigns to any element of $\Sigma^* \cup \Sigma^\omega$ a closed subinterval of $\mathbb{R}^\star$:

$$\begin{cases} [\![[]]\!] & = \mathbb{R}^\star \\ [\![\alpha_0\alpha_1\cdots\alpha_n]\!] & = \phi_{\alpha_0} \circ \phi_{\alpha_1} \cdots \circ (\phi_{\alpha_n}(\mathbb{R}^\star)) \\ [\![\alpha_0\alpha_1\cdots]\!] & = \bigcap_{n=0}^{\infty} [\![\alpha_0\alpha_1\cdots\alpha_n]\!]. \end{cases} \tag{5.1.1}$$

We call $(\Sigma, [\![\_]\!])$ an *IFS-representation*[6] for $\mathbb{R}^\star$ if

i) $\forall \sigma \in \Sigma^\omega \exists x \in \mathbb{R}^\star.\ [\![\sigma]\!] = \{x\}$

ii) $\forall x \in \mathbb{R}^\star \exists \sigma \in \Sigma^\omega.\ [\![\sigma]\!] = \{x\}$

[6] IFS is an abbreviation for 'iterated function systems' and comes form the literature of fractal theory [8].

An IFS-representation is *open* iff for for each $x \in (0,+\infty)$, there is $\alpha \in \Sigma$ s.t. $x \in \phi_\alpha((0,+\infty))$. One can prove that open IFS-representations are admissible representations.

We present an open IFS-representation $(\{\mathbf{L},\mathbf{R},\mathbf{M}\},[\![\_]\!])$ by defining the Möbius maps $\phi_{\mathbf{L}} = \left[\begin{smallmatrix}1 & 0\\ 1 & 1\end{smallmatrix}\right]$ and $\phi_{\mathbf{R}} = \left[\begin{smallmatrix}1 & 1\\ 0 & 1\end{smallmatrix}\right]$ and the map

$$\phi_{\mathbf{M}}(x) \;=\; \begin{cases} \frac{1}{2-x} & x \leq 1,\\ \frac{2x-1}{x} & 1 < x. \end{cases}$$

Obviously all Möbius maps are d-contraction. Moreover the map $\phi_{\mathbf{M}}$ — which is piecewise Möbius— is also a d-contraction. Therefore we can define $[\![\_]\!]$ as in (5.1.1). Consequently we can prove the following theorem:

Theorem 5.2 $(\{\mathbf{L},\mathbf{R},\mathbf{M}\},[\![\_]\!])$ *is an open IFS-representation.*

The proof of the above theorem and its application to extend the algorithms of Sect. 4 to get an implementation of exact arithmetic will be given in a coming work of the author. Moreover one can extend the representation to include all real numbers by adding d-contractions corresponding to the sign. The openness (i.e. admissibility) of this representation guarantees that all the usual field operations on real numbers are computable based on this representation. One important point that we want to emphasise, is that since we have $\phi'_{\mathbf{M}}(1) = 1$, our representation does not satisfy the conditions for the limitation theorem of [8]. Hence it remains open to classify the set of all computable functions based on this representation[7].

Clearly if we only consider the ternary finite sequences, we obtain another representation for rational numbers. In Figure II we show the new version of Stern–Brocot tree, a redundant one, in which every rational number has infinitely many representations. As wee see the new structure is not a tree anymore. In particular one can see that the following equalities hold:

$$\forall n\; \mathbf{M}^n\mathbf{R} = \mathbf{R}\mathbf{L}^n, \quad \mathbf{M}^n\mathbf{L} = \mathbf{L}\mathbf{R}^n, \quad \mathbf{M}^\infty = \mathbf{L}\mathbf{R}^\infty = \mathbf{R}\mathbf{L}^\infty$$

The ternary sequences ending in a run of **M**s corresponds to the non-reduced fraction. By using Lemma 3.2, it can be shown that the number of representations for each reduced fraction corresponds to the number of the length of the $\mathbb{N}$-fraction of that rational number; or equivalently the number of divisions in the Euclid's algorithm for calculating the gcd of the numerator and the denominator.

It is possible to build the redundant Stern–Brocot structure level by level. The process is similar to the construction of Stern–Brocot tree itself. The nth row consists of the nth row of Stern–Brocot tree plus the mediants every two consecutive elements of the nth row of the Stern–Brocot tree. These mediants are placed between their 'parents'. This symmetry seems to be a very strong reason for adopting this particular ternary representation based on Stern–Brocot tree. However, it is possible to consider alternative representations by changing the definition of $\phi_{\mathbf{M}}$ (e.g. to take a Möbius map instead of a piecewise Möbius map). In that case the corresponding Stern–Brocot structure will become more complex.

6 Conclusion and further work

In the present paper we showed that Stern–Brocot tree can be considered as a suitable approach to exact arithmetic on rational numbers. The main contributions of the present paper are the algorithms presented in Section 4. This algorithms of Sections 4.1 and 4.2 has been implemented in the lazy programming language Haskell and the Coq theorem prover [18]. As we mentioned earlier, the proof of theorem 4.11 (which is a special case of the proof of theorem 4.21), is also formalised in Coq. The algorithms and their formalisation in Coq are available online as contribution to the Coq contribution library at [13]. This formalisation is joint work with Yves Bertot and is explained in [14].

[7] With the terminology of [8] the above representation is a *slow* IFS-representation.

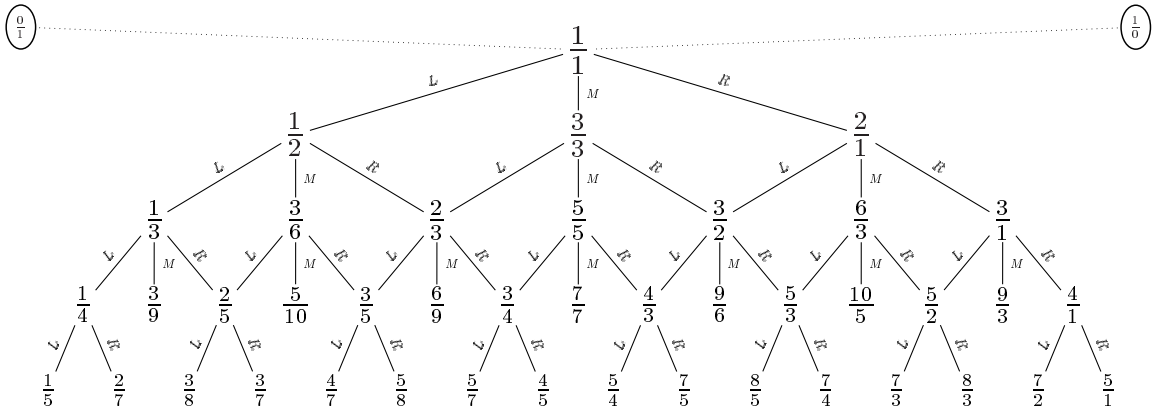

Figure II: the redundant Stern–Brocot structure

The author began the entire project when he attempted to verify the algorithms for exact real arithmetic using Coq theorem prover. After simplifying various representations based on continued fractions, the Stern–Brocot representations seemed to be the most suitable one to be formalised inside a theorem prover. The priority was to formally verify lazy algorithms for exact arithmetic. This may explain the inefficiency of the algorithm from a computational complexity point of view. Although the algorithms are relatively efficient when applied on rational numbers with small denominators. Some of the issues which were omitted in the present paper, such as well-orderings and termination of the recursive calls and the issues peculiar to the verification of the algorithms on a theorem prover are discussed in [14].
The proof of the results mentioned in Section 5 will be presented in a separate paper.

Further work is the project of formally verifying the algorithms on real arithmetic based on Stern–Brocot tree. For this the representation should be considered as a coinductive type in type theory. Coinductive types are types corresponding to infinite objects and using them the algorithms of exact real arithmetic can be formalised in a theorem prover like Coq.

Acknowledgement. The author wishes to thank Herman Geuvers and Henk Barendregt for fruitful discussions on the subject.

References

[1] Bruce P. Bates. *Self-Matching and Interleaving in Some Integer Sequences and the Gauss Map*. PhD thesis, University of Wollogong, 2001.

[2] Yves Bertot. Simple canonical representation of rational numbers. In Herman Geuvers and Fairouz Kamareddine, editors, *Proc. of the workshop on Mathematics, Logic and Computation, Eindhoven, MLC'03*, volume 85 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2003.

[3] Achille Brocot. Calcul des rouages par approximation, nouvelle méthode. *Revue chronométrique. Journal des horlogers, scientifique et pratique*, 3:186–194, 1861.

[4] Abbas Edalat and Peter John Potts. A new representation for exact real numbers. In Stephen Brookes and Michael Mislove, editors, *Mathematical Foundations of Progamming Semantics, Thirteenth Annual Conference (MFPS XIII), Carnegie Mellon University, Pittsburgh, PA, USA, March 23–26, 1997*, volume 6 of *entcs*. elsevier, 1998.

[5] Ralph W. Gosper. HAKMEM, Item 101 B. `http://www.inwap.com/pdp10/hbaker/hakmem/cf.html#item101b`, feb. 1972. MIT AI Laboratory Memo No.239.

[6] Ronald E. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[7] Brian Hayes. On the teeth of wheels. *American Scientist*, 88(4):296–300, july–august 2000.

[8] Michal Konečný. *Many-Valued Real Functions Computable by Finite Transducers using IFS-Representations*. PhD thesis, School of Computer Science, The University of Birmingham, oct 2000.

[9] Peter Kornerup and David Matula. An on-line arithmetic unit for bit-pipelined rational arithmetic. *Journal of Parallel and Distributed Computing*, 5:310–330, 1988.

[10] Peter Kornerup and David Matula. LCF: A lexicographic binary representation of the rationals. *Journal of Universal Computer Science*, 1(7):484–503, july 1995.

[11] Pierre Liardet and Pierre Stambul. Algebraic computations with continued fractions. *Journal of Number Theory*, 73:92–121, 1998.

[12] Valérie Ménissier-Morain. *Arithmétique exacte, conception, algorithmique et performances d'une implémentation informatique en précision arbitraire*. Thèse, Université Paris 7, dec 1994.

[13] Milad Niqui and Yves Bertot. `http://coqcvs.inria.fr/cgi-bin/cvswebcoq.cgi/contrib/Nijmegen/QArith/`, may 2003. Files under cvs version of Coq 7.4.

[14] Milad Niqui and Yves Bertot. QArith: Coq Formalisation of Lazy Rational Arithmetic. *submitted*, oct 2003.

[15] Peter J. Potts. *Exact Real Arithmetic using Möbius Transformations*. PhD thesis, Uiversity of London, Imperial College, jul 1998.

[16] George N. Raney. On continued fractions and finite automata. *Mathematische Annalen*, 206:265–283, 1973.

[17] Moritz Abraham Stern. Ueber eine zahlentheoretische Funktion. *Journal für die Reine und angewandte Mathematik*, 55:193–220, 1858.

[18] The Coq Development Team. *The Coq Proof Assistant Reference Manual, Version 7.4*. INRIA, `http://coq.inria.fr/doc/main.html`, feb 2003.

[19] Jean E. Vuillemin. Exact real computer arithmetic with continued fractions. *IEEE Transactions on Computers*, 39(8):1087–1105, aug 1990.

[20] Klaus Weihrauch. *Computable Analysis*. Springer-Verlag, Berlin Heidelberg, 2000. 285 pp.