Available online at www.sciencedirect.com

ScienceDirect

Journal of Discrete Algorithms 5 (2007) 356–379

JOURNAL OF DISCRETE ALGORITHMS

www.elsevier.com/locate/jda

Exact arithmetic on the Stern–Brocot tree

Milad Niqui *

Institute for Computing and Information Sciences, Radboud University Nijmegen, Toernooiveld 1, 6525 ED, Nijmegen, The Netherlands

Received 26 February 2004; accepted 6 March 2005

Available online 11 May 2006

Abstract

In this paper we present the Stern–Brocot tree as a basis for performing exact arithmetic on rational numbers. There exists an elegant binary representation for positive rational numbers based on this tree [Graham et al., Concrete Mathematics, 1994]. We will study this representation by investigating various algorithms to perform exact rational arithmetic using an adaptation of the homographic and the quadratic algorithms that were first proposed by Gosper for computing with continued fractions. We will show generalisations of homographic and quadratic algorithms to multilinear forms in n variables. Finally, we show an application of the algorithms for evaluating polynomials.

Keywords: Stern–Brocot; Exact; Rational; Homographic; Quadratic; Multilinear

1. Historical background

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 (also called: on-line, call-by-need, corecursive, etc.): 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 [12], 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 maps that is developed by Edalat and Potts [6, 23]. Edalat and Potts's work generalises earlier works by Gosper [7,8], Vuillemin [27] and Menissier-Morain [19]. Gosper, in his famous unpublished work [7], showed how to add, multiply, subtract and divide 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 lazy exact arithmetic on rational numbers.

* Tel.: +31 24 3652610; fax: +31 24 3652728.
E-mail addresses: M.Niqui@cs.ru.nl, milad@cs.kun.nl.

1570-8667/$ – see front matter
doi:10.1016/j.jda.2005.03.007

Later Vuillemin [27] adapted the algorithms to work for a redundant representation of real numbers using continued fractions. Kornerup and Matula [13] presented a binary version of Gosper's algorithm on a bit-serial arithmetic unit. Later they introduced redundancy in their representation and considered arithmetic units supporting such redundancies [15]. Their binary encoding of continued fraction expansion was a lexicographic one and they could use it to obtain a redundant representation for real numbers [14]. Lester [16] analyses the amount of redundancy needed in a continued fraction representation in order for it to be computationally suitable.

Our approach in the present paper is similar to the one by Kornerup and Matula. Our basis is the Stern–Brocot tree, a tree which was first discovered by 19th century German mathematician Moritz Abraham Stern and French clockmaker Achille Brocot [3,10,26]. The tree itself is a symmetric mathematical structure with remarkable algebraic and combinatorial properties. It was recently reintroduced by Graham et al. [9]. 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 [9] 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[1] of the regular continued fraction expansion of rational numbers and is presented in Raney [25]. 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 [17] 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 mathematical proofs of properties of rational 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 p_1/q_1 and p_2/q_2 we define the *mediant* of them to be the fraction $(p_1 + p_2)/(q_1 + q_2)$. We refer to p_1/q_1 and p_2/q_2 as *parents* of $(p_1 + p_2)/(q_1 + q_2)$. 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 $0/1$ and $1/0$, we proceed to construct the tree row by row. The first row is the mediant of the two initial pseudo-fractions, that is $(0 + 1)/(1 + 0) = 1/1$. We write this mediant in the middle of the initial pseudo-fractions. The second row consists of the mediant of $(0/1, 1/1)$ and the mediant of $(1/1, 1/0)$. We continue in this way and each time we choose two fractions that do not have anything vertically between them, and we place their mediant in a new row and vertically in the middle of the two parents. This construction is illustrated in Fig. 1. 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 [9, pp. 117–119] and [1].

Lemma 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.*

From Lemma 1(ii) it follows that the Stern–Brocot tree is an enumeration of the positive rational numbers. It has already been observed [9] 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 $1/1$ 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 $5/12$ 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 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 $13/8$. We call this *the Stern–Brocot representation* of positive rational numbers and throughout the rest of the paper we present algorithms for computations with this representation.

[1] Unary, in the sense that a partial denominator n of the continued fraction is denoted by a string of length n (cf. Lemma 9).

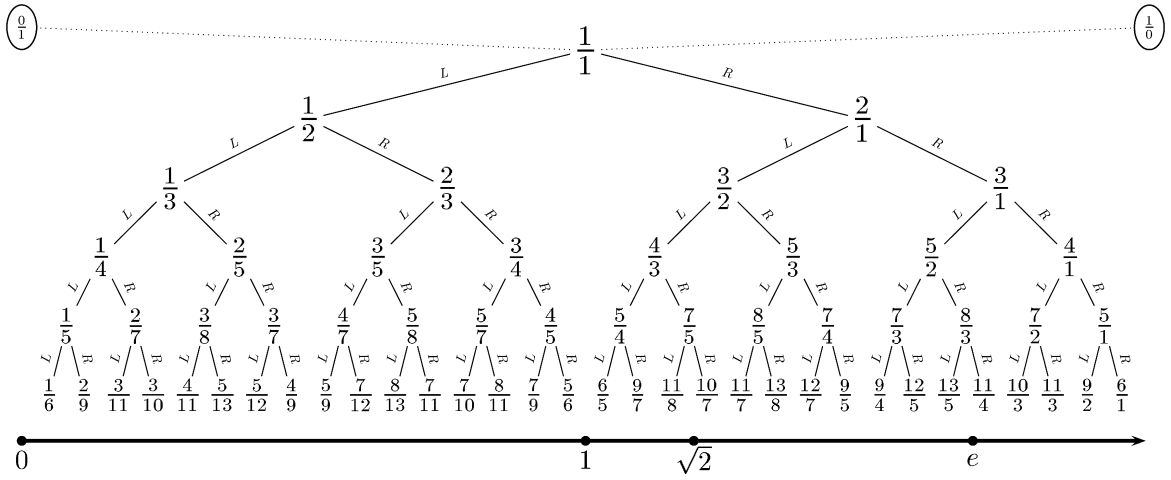

Fig. 1. The Stern–Brocot tree.

Remark 2. The binary nature of the Stern–Brocot tree and its step by step construction is reminiscent of the Conway's construction of the surreal numbers [4]. Each surreal number is approximated by a left set and a right set whose rôle is similar to that of the parents in the Stern–Brocot tree. The difference is that in Stern–Brocot tree each rational number p/q appears after finitely many steps of the construction, but in the tree of surreal numbers only dyadic rational numbers appear after finite steps. However, transfinite iteration of the construction method of the surreal numbers produces much more than all rational and real numbers. This transfinite nature can be seen in the way the arithmetic operation on the surreal numbers are evaluated. Therefore a straightforward formalisation of algorithms for computation on the surreal numbers needs a more powerful framework than most present programming languages and should best be done in a framework where higher order types are present, as it is done in *Coq* [18]. However, the algorithms that we present for the Stern–Brocot arithmetic can be implemented in ordinary programming languages.

First we fix some notations. Let $\Sigma = \{\alpha_0, \alpha_1, \ldots, \alpha_n\}$ be a finite set of symbols. By Σ^* we denote the set of finite sequences of elements of Σ. In order to simplify the notation we treat the elements of Σ^* as *strings*, i.e., we do not put any marker between two consecutive element of the sequence. Let $\sigma, \sigma' \in \Sigma^*$, then $\sigma\sigma'$ is the string concatenation of σ and σ'. By $\sigma(n)$ we denote the nth element of σ. If $\alpha \in \Sigma$ is a symbol and $\sigma \in \Sigma^*$ is a sequence, then $\alpha\sigma$ denotes the sequence obtained by prepending α to the beginning of σ. In this case we call α the *head* element of σ. We denote the empty sequence by [ ].

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

Definition 3. Let $\mathbf{SB} = \{\mathbf{L}, \mathbf{R}\}^*$ be the set of finite sequences generated from two letters **L** and **R**. We define the following injection from the set of positive rational numbers $\mathbb{Q}^+$ to **SB**:

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

For every $x \in \mathbb{Q}^+$ we call $\ulcorner x \urcorner \in \mathbf{SB}$ *the unsigned Stern–Brocot binary representation of* x. We shall call the elements of **SB** the Stern–Brocot binary sequences.

Note that we do not require $\gcd(m, n) = 1$. This is justified by the following important lemma, which links the Stern–Brocot tree and the Stern–Brocot representation.

Lemma 4. *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 [9, pp. 120–122]. □

Combining this lemma with the Lemma 1, as a corollary one can prove that $\ulcorner m/n \urcorner = \ulcorner km/kn \urcorner$; hence $\ulcorner \_ \urcorner$ is well-defined on the set of positive rational numbers.

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

Definition 5. Let $\sigma \in \mathbf{SB}$. We define the map $[\![\_]\!] : \mathbf{SB} \to \mathbb{Q}^+$ as:

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

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

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

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 7. 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**:

$$\ulcorner q \urcorner := \begin{cases} \mathbf{Zero} & \text{if } q = 0, \\ \mathbf{Pos} \ulcorner q \urcorner & \text{if } 0 < q, \\ \mathbf{Neg} \ulcorner (-q) \urcorner & \text{if } q < 0. \end{cases}$$

Thus the set **SSB** is isomorphic to $\mathbb{Q}$. We are interested in transferring the computations on $\mathbb{Q}$ to computations on **SSB**. In other words we seek a way of directly equipping the set **SSB** with algebraic operations (i.e., field structure) apart from the naïve way, which is decoding the representations using $[\![\_]\!]$, applying usual addition and multiplication on $\mathbb{Q}$ as a set of pairs of integers, and encoding back the result using $\ulcorner \_ \urcorner$.

2.1. Stern–Brocot tree and continued fractions

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

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

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

$$p = a_0 q + r_0, \quad 0 < r_0 < q$$

$$q = a_1 r_0 + r_1, \quad 0 < r_1 < r_0$$
$$r_0 = a_2 r_1 + r_2, \quad 0 < r_2 < r_1$$
$$\ldots$$
$$r_{n-2} = a_n r_{n-1} + 0,$$

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

It can be easily observed that in the $\mathbb{N}$-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 (a proof can be found in [9]).

Lemma 9. *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*, where* $i_0, j_n \geqslant 0$ *and the other* i_k*s and* j_k*s are greater than* 0*. Then we have*:

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

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

The relationship between the Stern–Brocot representation and Euclid's algorithm can be seen as a justification of the main property of Stern–Brocot tree expressed in Lemma 1. More specifically, by inspecting the above $\gcd'$ function, the Stern–Brocot representation can be seen as keeping track of the number of the times that numerator (respectively denominator) is larger than the denominator (respectively numerator) by outputting **L** (respectively **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 the canonical proofs of the fact they are irreducible.

2.2. *Why study this representation?*

In the remainder of the paper we are going to present algorithm for calculating arithmetic operations directly on this binary representation. The main motivation of our study of this representation is a theoretical one. Practically, the algorithms that we are going to present are not efficient. This is due to the fact that our suggested algorithms—working with essentially unary notation—give an exponential slow-down compared to straightforward algorithms operating with numerator and denominator represented in binary.

Our first motive is to the contribute to elegant theory of the Stern–Brocot tree. Many combinatorial properties of the Stern–Brocot tree are known [1,9], but our work adds the algebraic structure of a field—albeit computationally inefficient—to this tree. We view this algebraic structure as an important theoretical property and hence present the algorithms for field operations in this paper.

Our second motive, and the origin of the current work, is that the proof of correctness of these algorithms is easy to formalise in a proof assistant. Of course, the ease of formalisation in itself can not justify the formalisation of the algorithms. However, due to verbose and relatively tedious state of formal proofs, in formalising truly efficient and computationally practical numerical algorithms one has to take a step-by-step approach. The first step would be to formalise a theoretically important but computationally simple structure and build increasingly complex structures on top of that. For example, the computationally inefficient unary *Peano* representation for natural numbers is usually used inside proof assistants as a reference point for verifying more efficient algorithms of integer arithmetic. In our view Stern–Brocot representation serves as a starting and reference point for verification of more sophisticated algorithms that follow the same general shape, such as those by Gosper [7], Kornerup and Matula [15] Lester [16] and Edalat and Potts [23,24].

We give a more concrete example for our *ease of formalisation* argument. Recall that the Stern–Brocot representation encodes—in a unary way—all subtraction steps in the $\gcd'$ function. This allows us to refrain from using division in our algorithms. Instead we use subtraction. Although division is a powerful computational tool, compared to subtraction it is rather costly for proofs. This is mainly due to the fact division is a partial function; in a formal framework

(e.g. a proof assistant) the proof obligation for the fact that the divisor is not zero should be carried around. A formalised version of division has three arguments: the dividend, the divisor and the proof that the divisor is not zero. This is not the case with the subtraction when defined on integers. Hence, our emphasis lies on the use of subtraction instead of division and we follow this line in the definition of the algorithms in next section.

Finally, as a further incentive, in Section 4 we present a possible application of our algorithms for evaluating polynomials and rational functions that motivates the study of multilinear algorithm.

3. Exact rational arithmetic

In this section we present the lazy algorithms for computation with the Stern–Brocot representation. Lazy computation is a constructive interpretation of continuity.[2] The idea is that if we are computing continuous functions on sequences, it is possible to do this computation in a lazy manner, outputting partial information about the final result after having processed only initial segments of the input. A lazy algorithm on sequences usually consists of two steps:

(1) looking at initial segments of the input, the *absorption* step;
(2) outputting an initial segment of the output, the *emission* step.

An algorithm terminates when it emits the empty sequence. When there are several inputs, the algorithm also contains an *absorption strategy* to decide which input initial segment to absorb next.

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 the 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 fractional forms (cf. Section 3.3), which means by getting each bit of the input we can approximate the output; if our approximation is precise enough then we can firmly determine the 'next' element of the output. We continue in this way until we eventually run out of the input or arrive at a stage where the rest of output is independent of *any additional* input. This method was proposed by Gosper [7] for arithmetic on continued fractions but is easily applicable to a wide variety of problems that deal with computing a continuous function, including all elementary functions [23].

In this section we follow this pattern to devise our new algorithms for computation with the Stern–Brocot representation, directly on **SSB**. First we present the homographic and quadratic algorithms. Subsequently we generalise these algorithms to n variables.

3.1. Homographic algorithm

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

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

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.[3] Moreover, 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 $\sigma \in$ **SSB**. Thus we are given a finite signed binary sequence and four integer coefficients and we want to produce a finite signed binary sequence that is the Stern–Brocot representation of $h(x)$.

In order to present the algorithm we only need to assume $\sigma \in$ **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

[2] There are other aspects of laziness (e.g. laziness in the sense of sharing the reduction) that we do not consider in this paper.

[3] However, in the formal verification of algorithms, we should pay attention to this point. We do that by introducing *lazy proof obligations* [20, §3.3].

for computing $\ulcorner h(\llbracket\tau\rrbracket)\urcorner$ when $\tau \in \mathbf{SSB}$. This is simply because if $\llbracket\tau\rrbracket = -\llbracket\sigma\rrbracket$ then:

$$\ulcorner h_M(\llbracket\tau\rrbracket)\urcorner = \ulcorner \frac{a\llbracket\tau\rrbracket + b}{c\llbracket\tau\rrbracket + d}\urcorner = \ulcorner \frac{(-a)\llbracket\sigma\rrbracket + b}{(-c)\llbracket\sigma\rrbracket + d}\urcorner = \ulcorner h_{M'}(\llbracket\sigma\rrbracket)\urcorner \quad \text{where } M' = \begin{bmatrix} -a & b \\ -c & d \end{bmatrix}.$$

Hence we let $\sigma \in \mathbf{SB}$. First we determine the sign of $h_M(\llbracket\sigma\rrbracket)$. We call the algorithm that determines the sign of the output the *sign algorithm*. In the course of determining the sign, each time we absorb input the matrix of coefficients changes. Thus the algorithm that determines the sign should not only output the sign but also the new matrix of coefficient 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-bit* algorithm.
- We combine the sign algorithm and the output-bit algorithm to compute $h_M(\llbracket\sigma\rrbracket)$ for $\sigma \in \mathbf{SB}$.

The idea behind the sign algorithm is that we absorb as much input as needed to be able to determine the sign of each of numerator and denominator. This is because absorbing bits of an Stern–Brocot binary sequence σ will enable us to narrow down the size of the interval that σ may belong to. In fact, we use this idea in all algorithms of this section.

To compute the sign note that $0 < \llbracket\sigma\rrbracket$. This mean that the numerator and denominator have a parameter that is always positive. This eases the task of determining the sign of numerator and denominator: all we need to do is to check the sign of the two lines $ax + b$ and $cx + d$ for positive values of x. In other words, we have to check the sign of $-b/a$ and $-d/c$. We do this using some auxiliary functions. First recall that sgn is the following function on integers.

$$\operatorname{sgn} x := \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x = 0, \\ -1 & \text{if } x < 0. \end{cases}$$

As we discussed in Section 2.2 we are trying to avoid integer division. Hence 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 10. Let $m, n \in \mathbb{Z}$. We define the *sum of signs of m and n* to be $\mathsf{ssg}(m, n) = \operatorname{sgn} m + \operatorname{sgn} n$.

We are going to use $\mathsf{ssg}(a, b)$ and $\mathsf{ssg}(c, d)$ to determine the sign of $h_M(x)$ for $x > 0$ (parts (iii)–(iv) of the following lemma). This is possible because in general we are able to use $\mathsf{ssg}(m, n)$ to determine the sign of $mx + n$ for $x > 0$ (parts (i)–(ii) of the following lemma). The proof of the following lemma is immediate by an straightforward case analysis.

Lemma 11.

(i) $\mathsf{ssg}(m, n) > 0$ *then* $\forall x > 0$, $mx + n > 0$.
(ii) $\mathsf{ssg}(m, n) < 0$ *then* $\forall x > 0$, $mx + n < 0$.
(iii) $\mathsf{ssg}(a, b) \times \mathsf{ssg}(c, d) > 0$ *then* $\forall x > 0$, $(ax + b)/(cx + d) > 0$.
(iv) $\mathsf{ssg}(a, b) \times \mathsf{ssg}(c, d) < 0$ *then* $\forall x > 0$, $(ax + b)/(cx + d) < 0$.

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 a Stern–Brocot binary sequence $\sigma \in \mathbf{SB}$. Furthermore we use the following matrices—defined in terms of M—in the absorption steps.

$$M_{\mathbf{L}} := \begin{bmatrix} a+b & b \\ c+d & d \end{bmatrix}, \qquad M_{\mathbf{R}} := \begin{bmatrix} a & a+b \\ c & c+d \end{bmatrix}.$$

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 matrix of coefficients and the remaining part of the input binary sequence. The algorithm is written in the form of a recursive function $\mathcal{S}\_ : M_{2\times 2}(\mathbb{Z}) \times \mathbf{SB} \to \{0, +1, -1\} \times M_{2\times 2}(\mathbb{Z}) \times \mathbf{SB}$.

Homographic sign algorithm for $a, b, c, d \in \mathbb{Z}$, $\sigma \in \mathbf{SB}$

$$
\begin{cases}
\mathcal{S}_M([\,]) := \begin{cases} (0, M, [\,]) & \text{if } \mathsf{ssg}(a,b) = 0, \\ (+1, M, [\,]) & \text{else if } \mathsf{ssg}(a,b) = \mathsf{ssg}(c,d), \\ (-1, M, [\,]) & \text{otherwise.} \end{cases} \\
\mathcal{S}_M(\sigma) := \begin{cases} \text{if } b = 0 & \begin{cases} ((\operatorname{sgn} a)(\operatorname{sgn} c), M, \sigma) & \text{if } d = 0, \\ (\operatorname{sgn} a, M, \sigma) & \text{else if } \mathsf{ssg}(c,d) > 0, \\ (-\operatorname{sgn} a, M, \sigma) & \text{else if } \mathsf{ssg}(c,d) < 0, \\ \mathsf{absorb}_M(\sigma) & \text{otherwise.} \end{cases} \\ \text{if } b \neq 0 & \begin{cases} \text{if } d = 0 & \begin{cases} (\operatorname{sgn} c, M, \sigma) & \text{if } \mathsf{ssg}(a,b) > 0, \\ (-\operatorname{sgn} c, M, \sigma) & \text{else if } \mathsf{ssg}(a,b) < 0, \\ \mathsf{absorb}_M(\sigma) & \text{otherwise.} \end{cases} \\ \text{if } d \neq 0 & \begin{cases} (+1, M, \sigma) & \text{if } \mathsf{ssg}(a,b) \cdot \mathsf{ssg}(c,d) > 0, \\ (-1, M, \sigma) & \text{else if } \mathsf{ssg}(a,b) \cdot \mathsf{ssg}(c,d) < 0, \\ \mathsf{absorb}_M(\sigma) & \text{otherwise.} \end{cases} \end{cases} \end{cases}
\end{cases}
$$

The function $\mathsf{absorb\_} : M_{2\times 2}(\mathbb{Z}) \times \mathbf{SB} \to \{0, +1, -1\} \times M_{2\times 2}(\mathbb{Z}) \times \mathbf{SB}$ is defined by pattern matching on the head element of the input. This function is actually the recursive step in the $\mathcal{S}\_$ function and is also the absorption step.

$$
\begin{cases}
\mathsf{absorb}_M(\mathbf{L}\sigma') := \mathcal{S}_{M_{\mathbf{L}}}(\sigma'_1), \\
\mathsf{absorb}_M(\mathbf{R}\sigma') := \mathcal{S}_{M_{\mathbf{R}}}(\sigma'_1).
\end{cases}
$$

To understand the intuition behind this algorithm we first need to explain why the output is a triple. Recall that our goal is to pass the output from the sign algorithm to the output-bit algorithm. Thus the sign algorithm, after returning the sign (the first argument of the output triple) must return the part of the input that it has not used (the third argument of the output triple) and the matrix of coefficients, which is a modified version of the starting matrix of coefficients (the second argument of the output triple).

How the initial matrix is modified depends on which branch of the sign algorithm is followed. If the input sequence σ is empty then $[\![\sigma]\!] = 1$. Therefore we have to determine the sign of $h_M(1)$. As we see in the definition of the algorithm, in this case one can determine the sign of $(a+b)/(c+d)$ merely by checking $\mathsf{ssg}(a,b)$ and $\mathsf{ssg}(a,c)$. In this case we do not need to absorb any input, so the initial matrix of coefficients is returned unchanged. The third argument of the output (i.e., the 'rest' of the input) is trivially the empty sequence.

If the initial input sequence is not empty then we have to check whether b or d are zero. If both are zero then $h_M([\![\sigma]\!]) = a/c$ and we are done without absorbing any input. If $b = 0$ and $d \neq 0$ then $h_M([\![\sigma]\!]) = a[\![\sigma]\!]/(c[\![\sigma]\!] + d)$. In this case we use the Lemmas 11(i) and 11(ii) to determine the sign of $cx + d$. I.e., if $\mathsf{ssg}(c,d) \neq 0$ then independent of the actual value of $[\![\sigma]\!]$ and only by assuming that $[\![\sigma]\!] > 0$ we can firmly determine the sign of the denominator without absorbing any bits from σ. However, if $\mathsf{ssg}(c,d) = 0$ then we need more information about σ in order to be able to determine the sign. This is where we absorb information (i.e., bits) from σ by looking the head element of σ. If $\sigma = \mathbf{L}\sigma'$ then according to the Stern–Brocot tree we know that $[\![\sigma]\!]$ should be in the interval $(0, 1)$. We use this information indirectly, by replacing $[\![\sigma]\!]$ by $[\![\sigma']\!]/([\![\sigma']\!] + 1)$ in $h_{M_{\mathbf{L}}}([\![\sigma]\!])$ and obtaining $h_M([\![\sigma']\!])$. This replacement is possible because of the definition of $[\![\_]\!]$ (cf. Definition 5). This explains the definition of $M_{\mathbf{L}}$, which is the matrix of coefficients after the above replacement. After this replacement we continue to recursively determine the sign of $h_{M_{\mathbf{L}}}([\![\sigma']\!])$ by applying the sign algorithm with $M_{\mathbf{L}}$ and σ'. The other branches of the algorithm (and in particular the choice of matrix $M_{\mathbf{R}}$) can be justified in a similar fashion.

Having the output of the sign algorithm we now proceed to determine the bits of the output. 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 a Stern–Brocot binary sequence $\sigma \in \mathbf{SB}$. The structure of the output is similar to that of the $\ulcorner\_\urcorner$ algorithm: in each step we compare the numerator and denominator and we output $\mathbf{L}$ (respectively $\mathbf{R}$) if the numerator is less (respectively greater). Here, instead of numbers we compare the expressions $ax + b$ and $cx + d$ using the fact that $x > 0$. In order to facilitate this we define the following predicate.

Definition 12. Let $m, n, p, q \in \mathbb{Z}$. We define the predicate

$$\Xi_1(m, n, p, q) := (p \leqslant m \wedge q < n) \vee (p < m \wedge q \leqslant n).$$

The following trivial properties show why Ξ_1 is useful for the comparison of two line segments $ax+b$ and $cx+d$ when $x > 0$.

Lemma 13.

(i) *If* $\Xi_1(m,n,p,q)$ *then* $\forall x > 0,\ mx+n > px+q$.
(ii) *If* $\neg\Xi_1(m,n,p,q) \wedge \neg\Xi_1(p,q,m,n)$ *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 $\sigma \in \mathbf{SB}$. Furthermore, in the emission step, the algorithm uses the following matrices that are defined in terms of the matrix M.

$$M_N := \begin{bmatrix} a-c & b-d \\ c & d \end{bmatrix}, \qquad M_D := \begin{bmatrix} a & b \\ c-a & d-b \end{bmatrix}.$$

The output is a Stern–Brocot binary sequence $\sigma \in \mathbf{SB}$. The algorithm is written in the form of a recursive function $\mathcal{B}\_ : M_{2\times 2}(\mathbb{Z}) \times \mathbf{SB} \to \mathbf{SB}$.

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

$$\begin{cases} \mathcal{B}_M([\,]) := \ulcorner \frac{a+b}{c+d} \urcorner, \\ \mathcal{B}_M(\sigma) := \begin{cases} \mathbf{R}(\mathcal{B}_{M_N}(\sigma)) & \text{if } \Xi_1(a,b,c,d), \\ \mathbf{L}(\mathcal{B}_{M_D}(\sigma)) & \text{else if } \Xi_1(c,d,a,b), \\ \mathsf{absorb}'_M(\sigma) & \text{otherwise.} \end{cases} \end{cases}$$

The function $\mathsf{absorb}'\_ : T_{2\times(2\times 2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB} \to \mathbf{SB}$ is defined by pattern matching on the head element of input and is the absorption step of the above algorithm.

$$\begin{cases} \mathsf{absorb}'_M(\mathbf{L}\sigma') := \mathcal{B}_{M_{\mathbf{L}}}(\sigma'), \\ \mathsf{absorb}'_M(\mathbf{R}\sigma') := \mathcal{B}_{M_{\mathbf{R}}}(\sigma'). \end{cases}$$

As one can see, the intuition behind the output-bit algorithm is to mimic the encoding map $\ulcorner \_ \urcorner$ by comparing the numerator and the denominator of $h_M([\![\sigma]\!])$ using Lemma 13. If σ is the empty sequence then since $[\![\sigma]\!] = 1$, all that we need to do is to output the Stern–Brocot representation of $\ulcorner (a+b)/(c+d) \urcorner$. If σ is not empty then we try (by applying Lemma 13) to determine whether $ax+b$ is definitely greater or definitely smaller than $cx+d$ for all $x > 0$. If this is the case we can emit one bit of the output. For example assume $\Xi_1(a,b,c,d)$. Then we know that $a[\![\sigma]\!] + b > c[\![\sigma]\!] + d$, no matter what the exact value of $[\![\sigma]\!]$ is. Hence we know that $h_M([\![\sigma]\!])$ is a fraction whose numerator is larger than its denominator. Thus its Stern–Brocot representation should start with an $\mathbf{R}$; therefore, we output $\mathbf{R}$. Note that since $a[\![\sigma]\!] + b > c[\![\sigma]\!] + d$ by Definition 3 we have

$$\ulcorner \frac{a[\![\sigma]\!]+b}{c[\![\sigma]\!]+d} \urcorner = \mathbf{R} \ulcorner \frac{a[\![\sigma]\!]+b-(c[\![\sigma]\!]+d)}{c[\![\sigma]\!]+d} \urcorner. \tag{3.1}$$

In fact 'outputting' $\mathbf{R}$ means that we have used (3.1) to rewrite $\ulcorner h_M([\![\sigma]\!]) \urcorner$ as $\ulcorner h_{M_N}([\![\sigma]\!]) \urcorner$. This also explains the definition of M_N. In order to output further we make a recursive call to the output-bit algorithm with the modified matrix of coefficients M_N and the sequence σ. The definition of M_D and the other emission branches of the output-bit algorithm can be justified similarly. The absorption step of the output-bit algorithm is very similar to the one in the sign algorithm, i.e., we ask for more information by looking at the head element of σ and rewrite $h_M([\![\sigma]\!])$ accordingly. Here we use the same matrices $M_{\mathbf{L}}$ and $M_{\mathbf{R}}$ as in the sign algorithm to denote the modified matrix of coefficients that should be passed to the next recursive call.

An important issue is that in order for the output-bit algorithm to terminate 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 $(a+b)/(c+d)$. As a matter of fact, one can prove the following lemma.

Lemma 14. *Suppose* $a, b, c, d \geqslant 0$ *and* $a+b, c+d > 0$ *and* $M = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$. *Then the output-bit algorithm* $\mathcal{B}_M(\sigma)$ *terminates for all* $\sigma \in \mathbf{SB}$.

To prove this lemma one needs to define a well-ordering on the recursive arguments (in this case matrices and binary sequences) and apply the well-founded induction on this well-ordering. A formal proof of a variant of the above lemma has been formalised in the system *Coq* and can be found as *Coq* code in [21] (see also Lemma $\Psi^{\mathcal{B}}\_\mathtt{Wf}$ in [20, p. 82]). In fact in the type theory of *Coq* one can only compute with partial recursive functions by supplying a proof of their termination, i.e., one has to prove the above lemma in order to be able to compute with $\mathcal{B}$.

At this point we have to combine the sign and output-bit algorithms and obtain the homographic algorithm. 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. The algorithm assumes that the output of $\mathcal{S}_M(\sigma)$ 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 define the following matrices in terms of M_s.

$$M_s^- = \begin{bmatrix} -a_s & -b_s \\ -c_s & -d_s \end{bmatrix}, \qquad M_{s_1}^- = \begin{bmatrix} a_s & b_s \\ -c_s & -d_s \end{bmatrix}, \qquad M_{s_2}^- = \begin{bmatrix} -a_s & -b_s \\ c_s & d_s \end{bmatrix}.$$

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

Homographic algorithm for $a, b, c, d \in \mathbb{Z}, \sigma \in \mathbf{SB}$

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

The homographic algorithm mimics the function $\ulcorner \_ \urcorner$ as defined in Definition 7. Note that although $[\![\sigma]\!] > 0$ the sign of $h_M([\![\sigma]\!])$ is not necessarily positive (e.g. take $M = \left[\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$) and thus $h_M([\![\sigma]\!]) \in \mathbb{Q}$, as it was the case for the domain of $\ulcorner \_ \urcorner$ in Definition 7. The first thing that the homographic algorithm does, is to check whether the value of $h_M([\![\sigma]\!])$ is independent of σ. If so then the algorithm will evaluate the Stern–Brocot representation of the constant value of $h_M([\![\sigma]\!])$. Otherwise the algorithm will call the sign algorithm, which returns (s, M_s, σ_s).

We explain the case where s is negative (the case where s is positive is similar). In this case we know that $h_M([\![\sigma]\!])$—as an element of **SSB**—is of the form **Neg** τ where $\tau \in \mathbf{SSB}$. Computing τ is the task of the output-bit algorithm. The point here is that the output-bit algorithm should not be called with M_s as the initial matrix of coefficients, rather a modification of M_s. This has two reasons:

(1) The matrix of coefficients that is passed to the output-bit algorithm should satisfy the preconditions of Lemma 14, otherwise the output-bit algorithm would not terminate.
(2) The sign algorithm simply returns the sign without factorising this returned sign from the coefficients of M_s. Hence if s is negative then the coefficients of M_s should be modified to reflect this fact.

Returning $s = -1$ as sign of M_s means that $h_{M_s}(x) < 0$ for every $x > 0$. In particular $h_{M_s}(1) < 0$. Thus we use $h_{M_s}(1)$ to modify the signs of M_s; in order to satisfy the above two requirements we check whether $a_s + b_s$ is positive or negative (note that it can not be zero since $s = -1$). If it is positive then $c_s + d_s$ should be negative, otherwise $h_{M_s}(1)$ cannot be negative. Hence in this case we modify the coefficients of M_s and obtain $M_{s_1}^-$. The latter will then be passed to the output algorithm to determine the bits of τ.

First of all note that since $h_{M_s}(x) = -h_{M_{s_1}^-}(x)$, therefore the requirement (ii) is satisfied. To see that in this case (i.e., $a_s + b_s > 0$) $M_{s_1}^-$ satisfies the requirement (i) note that $a_s + b_s, (-c_s) + (-d_s) > 0$. The fact that $a_s, b_s \geqslant 0$ and $c_s, d_s \leqslant 0$ is entailed from the definition of $\mathcal{S}_M$ in the following way. Suppose $a_s < 0$. Since $a_s + b_s > 0$ we have $b_s > 0$. Thus $\mathsf{ssg}(a_s, b_s) = 0$. By examining the sign algorithm one observes that since $b_s \neq 0$ this algorithm only outputs the triple (s, M_s, σ_s) if $\mathsf{ssg}(a_s, b_s) \neq 0$. Hence our assumption that $a_s > 0$ leads to contradiction and therefore $a_s \geqslant 0$. Similarly one can prove that $b_s \geqslant 0$ and $c_s, d_s \leqslant 0$. Therefore all the preconditions of Lemma 14 hold.

One can similarly show that if $a_s + b_s < 0$, i.e., if $M_{s_2}^-$ is passed to the output-bit algorithm then the two requirements above hold.

We have explained the intuition behind the three algorithms that we presented in this section. The algorithms that we will present in Sections 3.2–3.3 are based on the same line of thought and the intuition behind them is the generalisation of the above intuitive ideas to the higher dimensions. Therefore, for the sake of brevity, we will not present a thorough explanation of the quadratic and multilinear algorithm.

The final point that we discuss in this section is the correctness of the homographic algorithm. Since the homographic algorithm is defined in terms of the sign algorithm and the output bit algorithm, we have to prove that all three algorithms are correct. Thus, we state the correctness of the homographic algorithm as the following theorem.

Theorem 15. *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* $\sigma \in \mathbf{SB}$.

(i) *Assume* $\mathcal{S}_M(\sigma) = (s, M_s, \sigma_s)$. *Then* $s = \operatorname{sgn} h_M([\![\sigma]\!])$.
(ii) *If* $a, b, c, d \geqslant 0$, $a + b > 0$ *and* $c + d > 0$ *then* $\ulcorner h_M([\![\sigma]\!])\urcorner = \mathcal{B}_M(\sigma)$.
(iii) $\ulcorner h_M([\![\sigma]\!])\urcorner = \mathcal{H}_M(\sigma)$.

The proof of this theorem has been formalised in the *Coq* proof assistant. The *Coq* code can found in the file `homographic_correctness.v` in [21]. This formalised proof is a special case of the proof of a more general fact that is stated in Theorem 32.

3.2. Quadratic algorithm

In this section we give a synchronised algorithm to compute addition and multiplication of two Stern–Brocot binary sequences. To do this we consider the following general *quadratic map*.

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

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 maps, we have a much more general algorithm and by taking the following special tensors for T we obtain the algorithms for basic arithmetic operations.

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

We define arithmetic operations on **SSB** as:

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

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

This way the operations $\oplus$, $\otimes$, $\ominus$ and $\oslash$ will satisfy their specifications. I.e.,

$$[\![\sigma_1 \oplus \sigma_2]\!] = [\![\sigma_q]\!] + [\![\sigma_2]\!], \qquad [\![\sigma_1 \ominus \sigma_2]\!] = [\![\sigma_q]\!] - [\![\sigma_2]\!],$$
$$[\![\sigma_1 \otimes \sigma_2]\!] = [\![\sigma_q]\!] \times [\![\sigma_2]\!], \qquad [\![\sigma_1 \oslash \sigma_2]\!] = \frac{[\![\sigma_q]\!]}{[\![\sigma_2]\!]}.$$

The quadratic algorithm is very similar to the homographic algorithm of the previous section. Once again it suffices to assume that $\sigma_1, \sigma_2 \in \mathbf{SB}$, i.e., σ_1 and σ_2 are the Stern–Brocot representation of *positive* rational numbers. The algorithm that 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. For instance if $[\![\tau_1]\!] = -[\![\sigma_1]\!]$ and $[\![\tau_2]\!] = [\![\sigma_2]\!]$ then:

$$\ulcorner q_T\big([\![\tau_1]\!], [\![\tau_2]\!]\big)\urcorner = \ulcorner \frac{a[\![\tau_1]\!][\![\tau_2]\!] + b[\![\tau_1]\!] + c[\![\tau_2]\!] + d}{e[\![\tau_1]\!][\![\tau_2]\!] + f[\![\tau_1]\!] + g[\![\tau_2]\!] + h} \urcorner$$

$$= \ulcorner \frac{(-a)[\![\sigma_1]\!][\![\sigma_2]\!] - b[\![\sigma_1]\!] + c[\![\sigma_2]\!] + d}{(-e)[\![\sigma_1]\!][\![\sigma_2]\!] - f[\![\sigma_1]\!] + g[\![\sigma_2]\!] + h} \urcorner$$

$$= \ulcorner q_{T'}([\![\sigma_1]\!], [\![\sigma_2]\!]) \urcorner \quad \text{where } T' = \begin{bmatrix} -a & -b & c & d \\ -e & -f & g & h \end{bmatrix}.$$

Hence we assume $\sigma_1, \sigma_2 \in \mathbf{SB}$ are given and we proceed exactly in the same way as in the previous section. First we determine the sign of $q_T([\![\sigma_1]\!], [\![\sigma_2]\!])$. Then we present the output-bit algorithm that for a positive numerator and denominator emits the output bits. Finally the quadratic algorithm combines the sign algorithm and the output-bit algorithm to compute $q_T([\![\sigma_1]\!], [\![\sigma_2]\!])$ for $\sigma_1, \sigma_2 \in \mathbf{SB}$.

In the quadratic sign algorithm, in contrast with the homographic sign algorithm, we absorb the bits from *both* inputs simultaneously. There are other possible absorption strategies but we will not consider this issue in this paper. For a discussion of strategies we refer the reader to [23, §11.5]. As in the case of homographic sign algorithm, the algorithm that determines the sign should not only output the sign but also the new tensor of coefficients and the remaining part of both inputs. First we need a definition similar to the Definition 10.

Definition 16. Let $m, n, p, q \in \mathbb{Z}$. We define the *sum of signs of m, n, p and q* to be $\mathsf{ssg}^{(2)}(m, n, p, q) := \operatorname{sgn} m + \operatorname{sgn} n + \operatorname{sgn} p + \operatorname{sgn} q$.

We shall use $\mathsf{ssg}^{(2)}(a, b, c, d)$ and $\mathsf{ssg}^{(2)}(e, f, g, h)$ to determine the sign of $q_T(x)$ for $x > 0$ (parts (iii)–(iv) of the following lemma). In doing so, we use parts (i)–(ii) of the following lemma to determine the sign of $mxy + nx + py + q$ for $x > 0$. The proof of the following lemma is immediate by an straightforward case analysis.

Lemma 17.

(i) *If* $\mathsf{ssg}^{(2)}(m, n, p, q) > 2$ *then* $\forall x, y > 0,\ mxy + nx + py + q > 0$.
(ii) *If* $\mathsf{ssg}^{(2)}(m, n, p, q) < -2$ *then* $\forall x, y > 0,\ mxy + nx + py + q < 0$.
(iii) *If* $\mathsf{ssg}^{(2)}(a, b, c, d) \times \mathsf{ssg}^{(2)}(e, f, g, h) > 8$ *then*

$$\forall x, y > 0, \quad \frac{axy + bx + cy + d}{exy + fx + gy + h} > 0.$$

(iv) *If* $\mathsf{ssg}^{(2)}(a, b, c, d) \times \mathsf{ssg}^{(2)}(e, f, g, h) < -8$ *then*

$$\forall x, y > 0, \quad \frac{axy + bx + cy + d}{exy + fx + gy + h} < 0.$$

The quadratic 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 $\sigma_1, \sigma_2 \in \mathbf{SB}$. In the absorption step (which is given as a separate function $\mathsf{absorb}^{(2)}$) we make use of the following tensors.

$$T_{\mathbf{LL}} = \begin{bmatrix} a+b+c+d & b+d & c+d & d \\ e+f+g+h & f+h & g+h & h \end{bmatrix}, \qquad T_{\mathbf{LR}} = \begin{bmatrix} a+c & a+b+c+d & c & c+d \\ e+g & e+f+g+h & g & g+h \end{bmatrix},$$

$$T_{\mathbf{RL}} = \begin{bmatrix} a+b & b & a+b+c+d & b+d \\ e+f & f & e+f+g+h & f+h \end{bmatrix}, \qquad T_{\mathbf{RR}} = \begin{bmatrix} a & a+b & a+c & a+b+c+d \\ e & e+f & e+g & e+f+g+h \end{bmatrix}.$$

The output is a quadruple $(s, T_s, \sigma_{1_s}, \sigma_{2_s})$ consisting of the sign, the new tensor of coefficients and the remaining parts of both inputs. The algorithm is written in the form of a recursive function $\underline{\mathcal{S}}^{(2)} : T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB} \to \{0, +1, -1\} \times T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB}$.

In the following let

$$A_1 = \begin{bmatrix} a+b & c+d \\ e+f & g+h \end{bmatrix}, \qquad A_2 = \begin{bmatrix} a+c & b+d \\ e+g & f+h \end{bmatrix},$$

$$S_{A_1}(\sigma_1) = (s_1, B_1, \sigma_{s_1}), \qquad S_{A_2}(\sigma_2) = (s_2, B_2, \sigma_{s_2}),$$

$$B_1 = \begin{bmatrix} a_{s_1} & b_{s_1} \\ c_{s_1} & d_{s_1} \end{bmatrix}, \qquad B_2 = \begin{bmatrix} a_{s_2} & b_{s_2} \\ c_{s_2} & d_{s_2} \end{bmatrix},$$

$$\overline{B_1} = \begin{bmatrix} 0 & a_{s_1} & 0 & b_{s_1} \\ 0 & c_{s_1} & 0 & d_{s_1} \end{bmatrix}, \qquad \overline{B_2} = \begin{bmatrix} 0 & 0 & a_{s_2} & b_{s_2} \\ 0 & 0 & c_{s_2} & d_{s_2} \end{bmatrix},$$
$$\varkappa_1 = \mathsf{ssg}^{(2)}(a,b,c,d), \qquad \varkappa_2 = \mathsf{ssg}^{(2)}(e,f,g,h).$$

Quadratic sign algorithm for $a,b,c,d,e,f,g,h \in \mathbb{Z}, \sigma_1, \sigma_2 \in \mathbf{SB}$

$$\begin{cases} \mathcal{S}^{(2)}_T(\sigma_1, [\,]) := (s_1, \overline{B_1}, \sigma_{s_1}, [\,]), \\ \mathcal{S}^{(2)}_T([\,], \sigma_2) := (s_2, \overline{B_2}, [\,], \sigma_{s_2}), \\ \mathcal{S}^{(2)}_T(\sigma_1, \sigma_2) := \begin{cases} \text{if } b = c = d = 0 & \begin{cases} ((\operatorname{sgn} a)(\operatorname{sgn} e), T, \sigma_1, \sigma_2) & \text{if } f = g = h = 0, \\ (\operatorname{sgn} a, T, \sigma_1, \sigma_2) & \text{else if } \varkappa_2 > 2, \\ (-\operatorname{sgn} a, T, \sigma_1, \sigma_2) & \text{else if } \varkappa_2 < -2, \\ \mathsf{absorb}^{(2)}_T(\sigma_1, \sigma_2) & \text{otherwise.} \end{cases} \\ \text{otherwise} & \begin{cases} \text{if } f = g = h = 0 & \begin{cases} (\operatorname{sgn} e, T, \sigma_1, \sigma_2) & \text{else if } \varkappa_1 > 2, \\ (-\operatorname{sgn} e, T, \sigma_1, \sigma_2) & \text{else if } \varkappa_1 < -2, \\ \mathsf{absorb}^{(2)}_T(\sigma_1, \sigma_2) & \text{otherwise.} \end{cases} \\ \text{otherwise} & \begin{cases} (+1, T, \sigma_1, \sigma_2) & \text{if } \varkappa_1 \cdot \varkappa_2 > 8, \\ (-1, T, \sigma_1, \sigma_2) & \text{else if } \varkappa_1 \cdot \varkappa_2 < -8, \\ \mathsf{absorb}^{(2)}_T(\sigma_1, \sigma_2) & \text{otherwise.} \end{cases} \end{cases} \end{cases} \end{cases}$$

The function $\mathsf{absorb}^{(2)} : T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB} \to \{0,+1,-1\} \times T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB}$ is similar to the function absorb that we used in defining the homographic sign algorithm and is used in the absorption step. Note that at each call to this function we absorb the head element of both inputs simultaneously.

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

Next we present the quadratic output-bit algorithm. First we need a generalisation of the Definition 12, which is used to facilitate the comparison of two surfaces, i.e., the numerator and the denominator.

Definition 18. Let $m,n,p,q,r,s,t,u \in \mathbb{Z}$. We define the predicate:

$$\begin{aligned} \Xi_2(m,n,p,q,r,s,t,u) := {} & (r \leqslant m \wedge s \leqslant n \wedge t \leqslant p \wedge u < q) \vee \\ & (r \leqslant m \wedge s \leqslant n \wedge t < p \wedge u \leqslant q) \vee \\ & (r \leqslant m \wedge s < n \wedge t \leqslant p \wedge u \leqslant q) \vee \\ & (r < m \wedge s \leqslant n \wedge t \leqslant p \wedge u \leqslant q). \end{aligned}$$

The following trivial property justifies the above definition.

Lemma 19.

(i) *If* $\Xi_2(m,n,p,q,r,s,t,u)$ *then*

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

(ii) *If* $\neg\Xi_2(m,n,p,q,r,s,t,u) \wedge \neg\Xi_2(m,n,p,q,r,s,t,u)$ *then*

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

We are ready to present the output-bit algorithm. The input to this algorithm is a tensor of coefficients and two Stern–Brocot binary sequences. In the emission steps we make use of the following tensors.

$$T_N = \begin{bmatrix} a-e & b-f & c-g & d-h \\ e & f & g & h \end{bmatrix}, \qquad T_D = \begin{bmatrix} a & b & c & d \\ e-a & f-b & g-c & h-d \end{bmatrix}.$$

The output is a Stern–Brocot binary sequence. The absorption step is given as a separate function $\mathsf{absorb}'^{(2)}$. The algorithm is given in the form of a recursive function $\mathcal{B}^{(2)}_{\_} : T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB} \to \mathbf{SB}$.

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

$$\begin{cases} \mathcal{B}^{(2)}_T(\sigma_1, [\,]) := \mathcal{B}_{A_1}(\sigma_1), \\ \mathcal{B}^{(2)}_T([\,], \sigma_2) := \mathcal{B}_{A_2}(\sigma_2), \\ \mathcal{B}^{(2)}_T(\sigma_1, \sigma_2) := \begin{cases} \mathbf{R}(\mathcal{B}^{(2)}_{T_N}(\sigma_1, \sigma_2)) & \text{if } \Xi_2(a,b,c,d,e,f,g,h), \\ \mathbf{L}(\mathcal{B}^{(2)}_{T_D}(\sigma_1, \sigma_2)) & \text{else if } \Xi_2(e,f,g,h,a,b,c,d), \\ \mathsf{absorb}'^{(2)}_T(\sigma_1, \sigma_2) & \text{otherwise.} \end{cases} \end{cases}$$

The function $\mathsf{absorb}'^{(2)}_{\_} : T_{2\times(2\times2)}(\mathbb{Z}) \times \mathbf{SB} \times \mathbf{SB} \to \mathbf{SB}$ is defined by pattern matching on the head element of input and is the absorption step of the above algorithm.

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

There is a precondition similar to Lemma 14 for the quadratic algorithm. For a proof see the *Coq* code in [21] and also Lemma $\Psi^{\mathcal{B}^{(2)}}$_Wf in [20, p. 84].

Lemma 20. *Suppose* $a, b, c, d, e, f, g, h \geqslant 0$ *and* $a + b + c + d, e + f + g + h > 0$ *and let* $T = \left[\begin{smallmatrix} a & b & c & d \\ e & f & g & h \end{smallmatrix}\right]$. *Then the quadratic output-bit algorithm* $\mathcal{B}^{(2)}_T(\sigma_1, \sigma_2)$ *terminates for all* $\sigma_1, \sigma_2 \in \mathbf{SB}$.

We have to combine the above algorithms and obtain the quadratic algorithm. The quadratic algorithm follows the same idea as the homographic algorithm. Recall that in the homographic algorithm we first tested whether the output is independent of any input, i.e., we checked whether the input matrix of coefficients is singular. We need to define similar notion of singularity for non-square matrices (in our case the $2 \times (2 \times 2)$ tensor). This is what the following predicate is intended for.

Definition 21. For $T = \left[\begin{smallmatrix} a & b & c & d \\ e & f & g & h \end{smallmatrix}\right]$ we define the predicate $\mathsf{same\_ratio}^{(2)}(T)$ as: $\mathsf{same\_ratio}^{(2)}(T) := af = be \wedge bg = cf \wedge ch = dg \wedge ag = ce \wedge ah = de \wedge bh = df$.

Finally we present the quadratic algorithm. 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 $\mathcal{S}^{(2)}_T(\sigma_1, \sigma_2)$ and we assume that the result is $(s, T_s, \sigma_{1_s}, \sigma_{2_s})$ where $T_s = \left[\begin{smallmatrix} a_s & b_s & c_s & d_s \\ e_s & f_s & g_s & h_s \end{smallmatrix}\right]$. Furthermore we use the following matrices to pass the right arguments to the output-bit algorithm.

$$T^-_s = \begin{bmatrix} -a_s & -b_s & -c_s & -d_s \\ -e_s & -f_s & -g_s & -h_s \end{bmatrix}, \qquad T^-_{s_1} = \begin{bmatrix} a_s & b_s & c_s & d_s \\ -e_s & -f_s & -g_s & -h_s \end{bmatrix},$$
$$T^-_{s_2} = \begin{bmatrix} -a_s & -b_s & -c_s & -d_s \\ e_s & f_s & g_s & h_s \end{bmatrix}.$$

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

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

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

To conclude this section we mention the following theorem, which shows the correctness of the above algorithms. The proof of this theorem has been formalised in the *Coq* proof assistant and can be found in the file `quadratic_correctness.v` in [21]. This formalised proof is a special case of the proof of a more general fact that is stated in Theorem 32.

Theorem 22. *Let* $q_T(x, y) = \frac{axy+bx+cy+d}{exy+fx+gy+h}$ *where* $T = \left[\begin{smallmatrix} a & b & c & d \\ e & f & g & h \end{smallmatrix}\right] \in T_{2\times(2\times2)}(\mathbb{Z})$. *Let* $\sigma_1, \sigma_2 \in \mathbf{SB}$.

(i) *Assume* $\mathcal{S}^{(2)}_T(\sigma_1,\sigma_2) = (s, T_s, \sigma_{1_s}, \sigma_{2_s})$. *Then* $s = \operatorname{sgn} q_T([\![\sigma_1]\!], [\![\sigma_2]\!])$.
(ii) *If* $a, b, c, d, e, f, g \geqslant 0, a + b + c + d > 0$ *and* $e + f + g + h > 0$ *then* $\ulcorner q_T([\![\sigma_1]\!], [\![\sigma_2]\!])\urcorner = \mathcal{B}^{(2)}_T(\sigma_1,\sigma_2)$.
(iii) $\ulcorner q_T([\![\sigma_1]\!], [\![\sigma_2]\!])\urcorner = \mathcal{Q}_T(\sigma_1,\sigma_2)$.

Note that the above theorem suffices to show the correctness of the field operations $\oplus,\ominus,\otimes$ and $\oslash$; because these are definable in terms of the quadratic map.

3.3. 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 $\mathcal{S}^{(2)}_T$, from $\mathcal{B}_M$ to $\mathcal{B}^{(2)}_T$ 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 15 and 22. 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 23. 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 $\mu = L_n/L'_n$ 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 the rational numbers are multilinear fractional forms of 0 variables, and homographic and quadratic maps are multilinear fractional forms of respectively 1 and 2 variables.

To generalise the quadratic algorithm we consider the following form.

$$\mu_{T_n}(x_1, \ldots, x_n) = \frac{L_n(x_1, \ldots, x_n)}{L'_n(x_1, \ldots, x_n)}, \quad L_n, L'_n \in \mathfrak{L}_n,$$

where T_n is a 2×2^n tensor:

$$T_n = \begin{bmatrix} a_{2^n} & a_{2^n-1} & \ldots & a_1 \\ b_{2^n} & b_{2^n-1} & \ldots & b_1 \end{bmatrix},$$

and $a_1, \ldots, a_{2^n}$ are coefficients of $L_n(x_1, \ldots, x_n)$ in ascending order (i.e., a_1 is the constant and a_{2^n} is the coefficient of the monomial with n variables). Similarly $b_1, \ldots, b_{2^n}$ are the coefficients of $L'_n(x_1, \ldots, x_n)$.

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 of coefficients whenever we have a multilinear fractional form.

We want to present the algorithm that given $\sigma_1, \ldots, \sigma_n \in \mathbf{SB}$ calculates the function $\ulcorner \mu_{T_n}([\![\sigma_1]\!], \ldots, [\![\sigma_n]\!]) \urcorner$. In order to be able to do that we need to fix some conventions.

First we present some notations for the *absorbed tensors*, which are the tensors of modified coefficients whenever we absorb from the input sequences. 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. There are 2^n different cases with respect to the head bit of each of the n input sequences being an $\mathbf{L}$ or an $\mathbf{R}$. We encode each of this 2^n different case by a sequence of length n consisting the head elements of each of the n input. We can enumerate these sequences as $\pi_1, \ldots, \pi_{2^n}$ by considering the lexicographic ordering on the n-element binary sequences of $\mathbf{L}$s and $\mathbf{R}$s. For example:

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

As an example, in the case of quadratic map we have 4 absorbed tensors, which were mentioned in Section 3.2 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 input sequences start with $\mathbf{L}$. We use the transformations that is implicit in Definition 5 to calculate the new coefficients. If we define $\phi(\mathbf{L}, x) = x/(x+1)$ 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:

$$\mu_{T_n}\big([\![\sigma_1]\!], \ldots, [\![\sigma_n]\!]\big) = \mu_{T_n}\big(\phi\big(\pi_i(1), [\![\sigma'_1]\!]\big), \ldots, \phi\big(\pi_i(n), [\![\sigma'_n]\!]\big)\big) = \mu_{T_{\pi_i}}\big([\![\sigma'_1]\!], \ldots, [\![\sigma'_n]\!]\big). \tag{3.2}$$

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 two (respectively four) tensors.

Furthermore we present the *emitted tensors*. Unlike the situation with the absorbed tensors, here, independent of n we always have two emitted tensors. That is because at any step in the output-bit algorithm we output $\mathbf{L}$ if the numerator is less than the denominator and we output $\mathbf{R}$ if denominator is less than the numerator (cf. Definition 3). If we can not decide which of the numerator or denominator is bigger then we should absorb more bits from the input sequences. Thus we denote the two emitted tensors by T_D (for the case of outputting $\mathbf{L}$) and T_N (for the case of outputting $\mathbf{R}$). Each time we output an $\mathbf{L}$ bit, we subtract the numerator from the denominator and each time we output an $\mathbf{R}$ we subtract the denominator from the numerator (cf. Definition 3). The emitted tensors will have the following shape.

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

Next tool that we need is a generalisation of the functions ssg and $\mathsf{ssg}^{(2)}$.

Definition 24. Let $m_1, \ldots, m_{2^n} \in \mathbb{Z}$. We define their *sum of signs* to be

$$\mathsf{ssg}^{(n)}(m_1, \ldots, m_{2^n}) = \sum_{i=1}^{2^n} \operatorname{sgn} m_i.$$

We use $\mathsf{ssg}^{(n)}$ as a criterion to determine the sign of a multilinear polynomial whose coefficients are $a_1, \ldots, a_{2^n}$. This is shown for simple cases in the Lemmas 11 and 17 and more generally in the following lemma.

Lemma 25.

(i) $\mathsf{ssg}^{(n)}(a_1, \ldots, a_{2^n}) > 2^n - 2$ *then* $\forall x_1, \ldots, x_n > 0,\ L_n(x_1, \ldots, x_n) > 0$.
(ii) $\mathsf{ssg}^{(n)}(a_1, \ldots, a_{2^n}) < -2^n + 2$ *then* $\forall x_1, \ldots, x_n > 0,\ L_n(x_1, \ldots, x_n) < 0$.
(iii) *If* $\mathsf{ssg}^{(n)}(a_1, \ldots, a_{2^n}) \times \mathsf{ssg}^{(n)}(b_1, \ldots, b_{2^n}) > 2^n(2^n - 2)$ *then* $\forall x_1, \ldots, x_n > 0,\ \mu_{T_n}(x_1, \ldots, x_n) > 0$.
(iv) *If* $\mathsf{ssg}^{(n)}(a_1, \ldots, a_{2^n}) \times \mathsf{ssg}^{(n)}(b_1, \ldots, b_{2^n}) < -2^n(2^n - 2)$ *then* $\forall x_1, \ldots, x_n > 0,\ \mu_{T_n}(x_1, \ldots, x_n) < 0$.

Proof.

(i) If $\mathsf{ssg}^{(n)}(a_1, \ldots, a_{2^n}) > 2^n - 2$ then

$$\mathsf{ssg}^{(n)}(a_1, \ldots, a_{2^n}) = 2^n - 1 \vee \mathsf{ssg}^{(n)}(a_1, \ldots, a_{2^n}) = 2^n.$$

Thus at least $2^n - 1$ of $a_1, \ldots, a_{2^n}$ 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) Let $X = \mathsf{ssg}^{(n)}(a_1, \ldots, a_{2^n})$ and $Y = \mathsf{ssg}^{(n)}(b_1, \ldots, b_{2^n})$ and assume

$$XY > 2^n\big(2^n - 2\big). \tag{3.3}$$

We prove that $|X|, |Y| > 2^n - 2$. Suppose on contrary that $|X|, |Y| < 2^n - 1$. Then $|XY| < (2^n - 1)^2$, i.e.,

$$|XY| \leqslant \big(2^n - 1\big)^2 - 1 = 2^n\big(2^n - 2\big),$$

which is in contradiction with (3.3). Thus we have proven that $|X|, |Y| > 2^n - 2$. Note that since $XY > 2^n(2^n - 2) \geqslant 0$ therefore X and Y are either both positive or both negative. Suppose the latter is the case (the former is similar). This means then $\mathsf{ssg}^{(n)}(a_1, \ldots, a_{2^n}) < -2^n + 2$ and $\mathsf{ssg}^{(n)}(b_1, \ldots, b_{2^n}) < -2^n + 2$. If $x_1, \ldots, x_n > 0$ by part (ii) we have

$$L_n(x_1, \ldots, x_n) < 0 \wedge L'_n(x_1, \ldots, x_n) < 0.$$

Therefore $\mu_n(x_1, \ldots, x_n) > 0$.
(iv) Similar to (iii). □

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

Definition 26. We shall denote the $2 \times 2^{n-1}$ tensor of coefficients of $\mu_n(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ by A_i. We assume that the outcome of

$$\mathcal{S}^{(n-1)}(A_i, \sigma_1, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n),$$

is of the following form:

$$(s_i, B_i, \sigma_{s_1}, \ldots, \sigma_{s_{i-1}}, \sigma_{s_{i+1}}, \ldots, \sigma_{s_n}),$$

where B_i is the new $2 \times 2^{n-1}$ tensor of coefficients. From B_i we construct the 2×2^n tensor $\overline{B_i}$. The tensor $\overline{B_i}$ is a tensor whose 2^{n-1} columns corresponding to the monomials *not* including x_i are the columns from B_i, and the other 2^{n-1} columns are 0. As an example the tensor $\overline{B_1}$ of Section 3.2 is obtained in this way.

When the input sequence σ_i $(1 \leqslant i \leqslant n)$ is the empty sequence, it means that one input sequence has vanished. Thus, in order to calculate μ_n we consider the coefficients of the monomials in which this vanishing variable occurs to be 0. That is to say, we consider $\mu_{T_n}(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ as another multilinear fractional form $\mu_{T'_n}(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ for some T'_n that can be obtained from T_n. This way, using the above notation, we output (as part of the output $n+2$-tuple) the tensor $\overline{B_i}$.

We mention some useful properties of the tensors A_i and $\overline{B_i}$. The proof is immediate by induction on n, the number of variables.

Lemma 27. *Let $\sigma_i = [\,]$ and A_i, B_i and $\overline{B_i}$ be defined as in Definition 26. Then*

(i) $\mu_{A_i}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = \mu_{T_n}(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$.
(ii) $\mu_{B_i}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = \mu_{\overline{B_i}}(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$.

In the following algorithm we use these shorthands.

$$\kappa_1 = \mathsf{ssg}^{(n)}(a_1, \ldots, a_{2^n}), \qquad \kappa_2 = \mathsf{ssg}^{(n)}(b_1, \ldots, b_{2^n}),$$
$$\kappa_{i^-} = \kappa_i - \left(2^n - 2\right), \qquad \kappa_{i^+} = \kappa_i + \left(2^n - 2\right), \qquad \left(i \in \{1, 2\}\right).$$

Multilinear sign algorithm for n variables

$$\begin{cases}
\mathcal{S}^{(n)}(T_n, \sigma_1, \ldots, \sigma_{i-1}, [\,], \sigma_{i+1}, \ldots, \sigma_n) := (s_i, \overline{B_i}, \sigma_{s_1}, \ldots \sigma_{s_{i-1}}, [\,], \sigma_{s_{i+1}}, \ldots, \sigma_{s_n}), \\
\mathcal{S}^{(n)}(T_n, \sigma_1, \ldots, \sigma_n) := \\
\quad \begin{cases}
\text{if } \forall i < 2^n,\ a_i = 0 & \begin{cases}
((\operatorname{sgn} a_{2^n})(\operatorname{sgn} b_{2^n}), T_n, \sigma_1, \ldots, \sigma_n) & \text{if } \forall i < 2^n,\ b_i = 0, \\
(\operatorname{sgn} a_{2^n}, T_n, \sigma_1, \ldots, \sigma_n) & \text{else if } \kappa_{2^-} > 0, \\
(-\operatorname{sgn} a_{2^n}, T_n, \sigma_1, \ldots, \sigma_n) & \text{else if } \kappa_{2^+} < 0, \\
\mathsf{absorb}^{(n)}_{T_n}(\sigma_1, \ldots, \sigma_n) & \text{otherwise.}
\end{cases} \\
\text{otherwise} & \begin{cases}
\text{if } \forall i < 2^n,\ b_i = 0 & \begin{cases}
(\operatorname{sgn} b_{2^n}, T_n, \sigma_1, \ldots, \sigma_n) & \text{else if } \kappa_{1^-} > 0, \\
(-\operatorname{sgn} b_{2^n}, T_n, \sigma_1, \ldots, \sigma_n) & \text{else if } \kappa_{1^+} < 0, \\
\mathsf{absorb}^{(n)}_{T_n}(\sigma_1, \ldots, \sigma_n) & \text{otherwise.}
\end{cases} \\
\text{otherwise} & \begin{cases}
(+1, T_n, \sigma_1, \ldots, \sigma_n) & \text{if } \kappa_1\kappa_2 > 2^n(2^n - 2), \\
(-1, T_n, \sigma_1, \ldots, \sigma_n) & \text{else if } \kappa_1\kappa_2 < -2^n(2^n - 2), \\
\mathsf{absorb}^{(n)}_{T_n}(\sigma_1, \ldots, \sigma_n) & \text{otherwise.}
\end{cases}
\end{cases}
\end{cases}
\end{cases}$$

The function $\mathsf{absorb}^{(n)} : T_{2\times 2^n}(\mathbb{Z}) \times \mathbf{SB}^n \to \{0, +1, -1\} \times T_{2\times 2^n}(\mathbb{Z}) \times \mathbf{SB}^n$ is the absorption step; it is defined by pattern matching on the head elements of the input sequences using the tensors T_{π_i} that were defined earlier in this section. Suppose the head elements of the n sequences form the sequence π_i. I.e., $\sigma_j = \pi_i(j)\sigma'_j$. Then

$$\mathsf{absorb}^{(n)}\left(T_n, \pi_i(1)\sigma'_1, \ldots, \pi_i(n)\sigma'_n\right) := \mathcal{S}^{(n)}(T_{\pi_i}, \sigma'_1, \ldots, \sigma'_n).$$

We turn our attention to the generalised form of the output-bit algorithm. First we generalise the Definitions 12 and 18 in order to obtain a predicate that can be used in the comparison of the numerator and the denominator. This predicate tries to determine whether one of numerator or denominator is greater than the other one for all $x_1, \ldots, x_n > 0$. In order to achieve this, it compares the coefficients of the identical monomials.

Definition 28. Let $p_1, \ldots, p_{2^n}, q_1, \ldots, q_{2^n} \in \mathbb{Z}$. We define the predicate:

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

Again we state those properties of this predicate that we use in the algorithm. The proofs are trivial.

Lemma 29.

(i) *If* $\Xi_n(a_1, \ldots, a_{2^n}; b_1, \ldots, b_{2^n})$ *then*

$$\forall x_1, \ldots, x_n > 0, \quad L_n(x_1, \ldots, x_n) > L'_n(x_1, \ldots, x_n).$$

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

$$\exists x_1, \ldots, x_n > 0, \quad L_n(x_1, \ldots, x_n) = L'_n(x_1, \ldots, x_n).$$

In the multilinear output-bit algorithm the input is the tensor T_n and the n Stern–Brocot binary sequences $\sigma_1, \ldots, \sigma_n \in \mathbf{SB}$. We present the algorithm as a recursive function $\mathcal{B}^{(n)} : T_{2\times 2^n}(\mathbb{Z}) \times \mathbf{SB}^n \longrightarrow \mathbf{SB}$. The recursion is on n as well as the input sequences and the tensor of coefficients. In order for this recursion to be provably terminating we need to define a well-ordering on the set of tensors. This is always possible (see the discussion before Theorem 32 and [22]).

Multilinear output-bit algorithm for n variables

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

Here the function $\mathsf{absorb}'^{(n)} : T_{2\times 2^n}(\mathbb{Z}) \times \mathbf{SB}^n \to \mathbf{SB}$ is defined by pattern matching on the head element of input similar to the above function $\mathsf{absorb}^{(n)}$.

$$\mathsf{absorb}'^{(n)}\big(T_n, \pi_i(1)\sigma'_1, \ldots, \pi_i(n)\sigma'_n\big) := \mathcal{B}^{(n)}(T_{\pi_i}, \sigma'_1, \ldots, \sigma'_n).$$

In the multilinear output-bit algorithm, if one of the input sequences is the empty sequences, we need to call the function $\mathcal{B}^{(n-1)}$ and perform the recursion on n. The preconditions that we had in Lemmas 14 and 20 for the base cases $n = 1$ and $n = 2$ will mount to the case of arbitrary n. Thus if for some $1 \leqslant i \leqslant n$ the input sequence σ_i is the empty sequence we need to require that for every $1 \leqslant i \leqslant 2^n$, $a_i, b_i \geqslant 0$ and that $L_n(1, 1, \ldots, 1) = \sum_{i=1}^{2^n} a_i > 0$ and $L'_n(1, 1, \ldots, 1) = \sum_{i=1}^{2^n} b_i > 0$. Note that in the case when one of the input sequences is empty we have a recursive step on n with a different tensor of coefficients. Furthermore, in the absorption step we have a recursive step with as argument a suitably chosen absorbed tensor. Hence we need the following lemma, which states that the preconditions are preserved during the recursive step.

Lemma 30.

(i) *For a tensor* $T \in T_{2\times 2^n}(\mathbb{Z})$ *by* $\mathsf{row}_1(T)$ *we mean the sum of the elements in the first row of the tensor; by* $\mathsf{row}_2(T)$ *we mean the sum of the elements in the second row of the tensor. Suppose* $\mathsf{row}_1(T_n), \mathsf{row}_2(T_n) > 0$. *Then* $\mathsf{row}_k(A_i) > 0$ $(k = 1, 2)$.

(ii) *Suppose for every* $1 \leqslant i \leqslant 2^n$, $a_i, b_i \geqslant 0$ *and let* $T_{\pi_i} = \left[\begin{smallmatrix} c_{2^n} & \cdots & c_1 \\ d_{2^n} & \cdots & d_1 \end{smallmatrix}\right]$. *Then for every* $1 \leqslant i \leqslant 2^n$, $c_i, d_i \geqslant 0$.

Proof.

(i) According to Lemma 27(i) A_i is obtained from T_n by simplifying $\mu_{T_n}(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$. By induction on n one can easily see that during the simplification $\mathsf{row}_k(T_n)$ remains equal to $\mathsf{row}_k(A_i)$.

(ii) Let a_i be the coefficient of the monomial $x_{i_1}x_{i_2}\ldots x_{i_j}$ in the numerator of μ_{T_n}. Recall that $\phi(\mathbf{L}, x) = x/(x+1)$ and $\phi(\mathbf{R}, x) = x+1$. From the definition of T_{π_i} it follows that c_i is obtained by (1) replacing $\phi(\pi_i(i_1), x_{i_1})$ in all monomials in T_n involving x_{i_1} $(1 \leqslant i \leqslant j)$, (2) simplifying the result and (3) adding up the coefficients of the monomial $x_{i_1}x_{i_2}\ldots x_{i_j}$. Since all the coefficients of T_n are positive and during the above simplification no negative factor enters the computation therefore $c_i \geqslant 0$. Similarly one can prove $d_i \geqslant 0$. □

The above lemma can be used to prove a result similar to Lemmas 14 and 20, namely that the multilinear output-bit algorithm terminates if all the coefficients are nonnegative and $\mathsf{row}_2(A_i), \mathsf{row}_2(A_i) > 0$ (cf. the statement of Theorem 32(ii)).

Finally, combining the above two algorithms we present the algorithm to compute the multilinear fractional form μ_n. Given the tensors T_n of the coefficients and the n Stern–Brocot binary sequences this algorithm will calculate $\ulcorner \mu_{T_n}([\![\sigma_1]\!], \ldots, [\![\sigma_n]\!]) \urcorner$. 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. As it was the case with the homographic and quadratic algorithm we use the outcome of the sign algorithm to correct the signs of the coefficients to satisfy the precondition. Thus assuming that $\mathcal{S}^{(n)}(T_n, \sigma_1, \ldots, \sigma_n) = (s, T_{n_s}, \sigma_{1_s}, \ldots, \sigma_{n_s})$, and that $T_{n_s} = \left[\begin{smallmatrix} a_{s_{2^n}} & a_{s_{2^n-1}} & \ldots & a_{s_1} \\ b_{s_{2^n}} & b_{s_{2^n-1}} & \ldots & b_{s_1} \end{smallmatrix}\right]$ 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 $T^-_{n_{s_1}}$ (respectively $T^-_{n_{s_2}}$) we negate only the elements in the second row (respectively first row) of T_{n_s} (cf. T^-_s, $T^-_{s_1}$ and $T^-_{s_2}$ in Section 3.2).

We also need a further generalisation of the notion of singularity for tensors; this predicate will tackle the case where output is independent of the input (cf. Definition 21).

Definition 31. For $T_n = \left[\begin{smallmatrix} a_{2^n} & a_{2^n-1} & \ldots & a_1 \\ b_{2^n} & b_{2^n-1} & \ldots & b_1 \end{smallmatrix}\right]$ we define the following predicate.

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

The multilinear algorithm has the form of a function $\mathcal{M}^{(n)} : T_{2\times 2^n}(\mathbb{Z}) \times \mathbf{SB}^n \to \mathbf{SSB}$. Note that there are no recursive calls in this function.

Multilinear algorithm for n variables

$$\begin{cases} \mathcal{M}^{(n)}(T_n, \sigma_1, \ldots, \sigma_n) := \\ \quad \begin{cases} \text{if } \mathsf{same\_ratio}^{(n)}(T_n) & \{\ulcorner \frac{a_i}{b_i} \urcorner \quad \text{for the first } 1 \leqslant i \leqslant 2^n \text{ s.t. } b_i \neq 0. \\ \text{otherwise} & \begin{cases} \mathbf{Zero} & \text{if } s = 0, \\ \text{if } s = 1 & \begin{cases} \mathbf{Pos}\ \mathcal{B}^{(n)}(T_{n_s}, \sigma_{1_s}, \ldots, \sigma_{n_s}) & \text{if } \mathsf{row}_1(T_{n_s}) > 0, \\ \mathbf{Pos}\ \mathcal{B}^{(n)}(T^-_{n_s}, \sigma_{1_s}, \ldots, \sigma_{n_s}) & \text{otherwise.} \end{cases} \\ \text{if } s = -1 & \begin{cases} \mathbf{Neg}\ \mathcal{B}^{(n)}(T^-_{n_{s_1}}, \sigma_{1_s}, \ldots, \sigma_{n_s}) & \text{if } \mathsf{row}_1(T_{n_s}) > 0, \\ \mathbf{Neg}\ \mathcal{B}^{(n)}(T^-_{n_{s_2}}, \sigma_{1_s}, \ldots, \sigma_{n_s}) & \text{otherwise.} \end{cases} \end{cases} \end{cases} \end{cases}$$

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

(1) Recursion on natural numbers. This leads to the 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) ordering of sequences according to their length. According to this well-ordering, the new input sequences (considered as an n-tuple) after an absorption step should be considered less than the

n-tuple of input sequences before the absorption. Therefore we use this induction scheme to prove the properties that are invariant during the absorption steps.

(3) Recursion on the structure of the tensor of 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 coefficients before the emission. Hence we use this induction scheme wherever our algorithm contains an emission step (e.g. 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 orders 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.

Here we state the correctness as a theorem. The proof involves a lengthy combination of the above three induction principles and multiple case analysis according to the branching of the algorithms. A detailed sketch of the proof, taking care of nontrivial parts, can be found in [20, §2.4].

Theorem 32 *(Correctness of the multilinear algorithm). Let*

$$\mu_{T_n}(x_1, \ldots, x_n) = \frac{L_n(x_1, \ldots, x_n)}{L'_n(x_1, \ldots, x_n)}$$

where $T_n = \left[\begin{smallmatrix} a_{2^n} & a_{2^n-1} & \ldots & a_1 \\ b_{2^n} & b_{2^n-1} & \ldots & b_1 \end{smallmatrix}\right] \in T_{2\times 2^n}(\mathbb{Z})$.

Let $\sigma_1, \ldots, \sigma_n \in \mathbf{SB}$.

(i) *Assume* $\mathcal{S}^{(n)}(T_n, \sigma_1, \ldots, \sigma_n) = (s, T_{n_s}, \sigma_{1_s}, \ldots, \sigma_{n_s})$. *Then*

$$s = \operatorname{sgn} \mu_{T_n}\big([\![\sigma_1]\!], \ldots, [\![\sigma_n]\!]\big).$$

(ii) *If* $\forall 1 \leqslant i \leqslant 2^n$, $a_i, b_i \geqslant 0$, $\mathsf{row}_1(T_n) > 0$ *and* $\mathsf{row}_2(T_n) > 0$. *Then*

$$\ulcorner \mu_{T_n}\big([\![\sigma_1]\!], \ldots, [\![\sigma_n]\!]\big) \urcorner = \mathcal{B}^{(n)}(T_n, \sigma_1, \ldots, \sigma_n).$$

(iii) $\ulcorner \mu_{T_n}([\![\sigma_1]\!], \ldots, [\![\sigma_n]\!]) \urcorner = \mathcal{M}^{(n)}(T_n, \sigma_1, \ldots, \sigma_n)$.

The above theorem is a generalisation of the facts stated in Theorems 15 and 22. For the cases $n = 1, 2$ (the homographic and quadratic algorithms), the proof has been formalised in *Coq* proof assistant (see Section 5 and [22]).

4. Polynomials

In the previous section we presented a method to compute the field operations directly on the Stern–Brocot representation of two rational numbers. In this section we discuss how one can use homographic, quadratic and multilinear algorithms to evaluate polynomials on Stern–Brocot representation. This means that, given a polynomial $P(x)$ with integer coefficients and in one variable, and given a Stern–Brocot binary sequence σ, we want to compute $\ulcorner P([\![\sigma]\!]) \urcorner$. We call this the *evaluation* of P at σ.

Let $P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$ be a polynomial of degree $n \geqslant 2$. The idea is to express the given polynomial as composition of multilinear fractional forms. In general there are several ways to do this. One way—which is similar to the usual *Horner* method for evaluating polynomials—consist of considering P_n as a quadratic map

$$q_0\big(x, P_{n-1}(x)\big) = \frac{1 \cdot x P_{n-1}(x) + 0 \cdot x + 0 \cdot P_{n-1}(x) + a_0}{0 \cdot x P_{n-1}(x) + 0 \cdot x + 0 \cdot P_{n-1}(x) + 1},$$

where $P_{n-1}(x) = a_n x^{n-1} + a_{n-1}x^{n-2} + \cdots + a_1$. Repeating this process for P_{n-1} it follows that we can write P_n as

$$P_n(x) = q_0\big(x, q_1\big(x, \ldots, q_{n-2}\big(x, h(x)\big)\big)\big), \tag{4.1}$$

where

$$h(x) = \frac{a_n x + a_{n-1}}{0.x + 1}.$$

According to (4.1) we have

$$\ulcorner P_n\big([\![\sigma]\!]\big)\urcorner = \mathcal{Q}_{\mathsf{Q}_0}\big(\sigma, \mathcal{Q}_{\mathsf{Q}_1}\big(\sigma, \ldots, \mathcal{Q}_{\mathsf{Q}_{n-2}}\big(\sigma, \mathcal{H}_{\mathsf{M}}(\sigma)\big)\big)\big);$$

where

$$\mathsf{Q}_i := \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathsf{M} := \begin{bmatrix} a_n & a_{n-1} \\ 0 & 1 \end{bmatrix}.$$

Clearly, in this case evaluation of P at σ requires $n-1$ calls to quadratic algorithm and one call to homographic algorithm.

The above method shows how one can use a composition of quadratic maps to evaluate a polynomial. We can generalise the above method by replacing q_0 with a multilinear fractional form μ_0 of k_0 variables for some $2 \leqslant k_0 \leqslant n$. On that account, repeating the procedure as wee did in the above method, we can write (4.1) as

$$P_n(x) = \mu_0\Big(\underbrace{x, \ldots, x}_{k_0-1 \text{ times}}, \mu_1\big(\underbrace{x, \ldots, x}_{k_1-1 \text{ times}}, \ldots, \mu_j(\underbrace{x, \ldots, x}_{k_j \text{ times}})\big)\Big),$$

where for all $i \leqslant j$ each μ_i is a multilinear fractional form of k_i variables with $2 \leqslant k_i \leqslant n$ and $1 \leqslant k_j$.[4] Thus we have

$$\ulcorner P_n\big([\![\sigma]\!]\big)\urcorner = \mathcal{M}^{(k_0)}_{\mathsf{T}_{k_0}}\Big(\underbrace{\sigma, \ldots, \sigma}_{k_0-1 \text{ times}}, \mathcal{M}^{(k_1)}_{\mathsf{T}_{k_1}}\big(\underbrace{\sigma, \ldots, \sigma}_{k_1-1 \text{ times}}, \ldots, \mathcal{M}^{(k_j)}_{\mathsf{T}_{k_j}}(\underbrace{\sigma, \ldots, \sigma}_{k_j \text{ times}})\big)\Big);$$

where for $i < j$

$$\mathsf{T}_{k_i} = \begin{bmatrix} 1 & \underbrace{0 \ldots 0}_{\binom{k_i}{k_i-1} \text{ times}} & a'_{k_i} & \underbrace{0 \ldots 0}_{\binom{k_i}{k_i-2}-1 \text{ times}} & \ldots & a'_3 & \underbrace{0 \ldots 0}_{\binom{k_i}{1}-1 \text{ times}} & a'_2 \\ 0 & & \ldots & & \ldots & & 0 & 1 \end{bmatrix},$$

and $a'_l = \sum_{t=0}^{i-1} k_t + l - (i+2)$. Moreover

$$\mathsf{T}_{k_j} = \begin{bmatrix} a_n & a_{n-1} & \underbrace{0 \ldots 0}_{\binom{k_j}{k_j-1}-1 \text{ times}} & a_{n-2} & \underbrace{0 \ldots 0}_{\binom{k_j}{k_j-2}-1 \text{ times}} & \ldots & a_{n-k_j} \\ 0 & & \ldots & & \ldots & 0 & 1 \end{bmatrix}.$$

Hence we can use the composition of multilinear algorithm to evaluate polynomials with integer coefficients.[5] Obviously, the complexity of the method depends on the complexity of multilinear algorithm. But it also depends on the choice of k_is (i.e., which composition of multilinear algorithms we consider). It is also possible to first factorise P_n, apply the above method to each factor, and finally multiply the factors by applying the multilinear algorithm on them.

In this paper we do not consider complexity of the algorithms of Section 3. However, the complexity of homographic algorithm is derivable from the methods that we present in [11]. The question of complexity of the multilinear algorithm for n variables seems to be a difficult problem.

Finally, a more general and more useful application of the multilinear algorithm would be the evaluation of the *rational functions*. Rational functions are elements of the field of fractions of the ring of polynomials over $\mathbb{Z}$. They can be written as a composition of multilinear fractional forms. However finding the optimal composition is a very difficult problem. The less complex problem of decomposing rational functions into an optimal composition of quadratic maps has already been posed as an open problem by Gosper [7, Item 101B]; but there seem to be no solution even in this simpler form.

[4] In fact, it is evident that $k_i \leqslant n + i - \sum_{t=0}^{i-1} k_t$ and that $\sum_{t=0}^{j} k_t = n + j$.

[5] Adaptation of the method for polynomials with rational coefficients is trivial.

5. Conclusion and further work

The main contributions of the present paper are the homographic, quadratic and multilinear algorithms presented in Sections 3.1–3.3. The underlying idea of the paper, the use of homographic and quadratic maps to lazily compute the arithmetic operation, is due to Gosper [7] where he presents them for continued fractions. The adaptation of this idea for use with the Stern–Brocot representation is nontrivial and can be considered as a novel contribution of the present paper. More specifically our treatment of sign and output-bit algorithm results in a more modular view of the main algorithms and eases the proof of the correctness. The generalisation of the homographic and quadratic algorithm to obtain the multilinear algorithm is another contribution of this paper. The suggested application of the multilinear algorithm for evaluating polynomials, which has its origin in an open problem posed by Gosper [7, Item 101B], is a further contribution of this paper.

The algorithms of Sections 3.1–3.2 have been implemented in the lazy programming language Haskell and formalised using the *Coq* proof assistant [5]. As we mentioned earlier the proofs of Theorems 22 and 15, which are special cases of the proof of Theorem 32, are also formalised in *Coq*. The algorithms and their formalisation in *Coq* are available on-line [21]. This formalisation is joint work with Yves Bertot and is explained in [22].

The author began the entire project when he attempted to verify the algorithms for exact real arithmetic using *Coq* proof assistant. 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 in order to assess the strength of the machinery of the proof assistant for tackling more efficient algorithms. This—in addition to the obvious theoretical interests—may explain the choice of this inefficient algorithms for formalisation. Although the algorithms are relatively efficient when applied on rational numbers with small denominators.

Some of the issues that 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 [22]. Some of the general issues regarding the extension of the representation to a representation for real numbers, and issues concerning the redundancy and the complexity of the representation can be found in [11,20].

One direction for further work is to consider alternative redundant representation for rational numbers based on other variants of gcd algorithm. These algorithms should result in more efficient versions of homographic and quadratic algorithm. Some work in this direction can be found in [20, §2.6]. Another possible direction would be 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 formal framework like *Coq*.

Acknowledgements

The present work was partially supported by the Netherlands Organisation for Scientific Research (NWO). The author wishes to thank Herman Geuvers and Henk Barendregt for fruitful discussions on the subject and the anonymous referees for their insightful comments and useful suggestions that helped in improving the article.

Appendix A. Glossary of notations

ssg	Sum of signs of m and n
$\mathcal{S}$	The homographic sign algorithm
absorb	The homographic sign algorithm (absorption)
Ξ_1	Predicate for comparing two line segments
$\mathcal{B}$	The homographic output-bit algorithm
absorb'	The homographic output-bit algorithm (absorption)
$\mathcal{H}$	The homographic algorithm
$\mathsf{ssg}^{(2)}$	Sum of signs of m, n, p and q
$\mathcal{S}^{(2)}$	The quadratic sign algorithm
$\mathsf{absorb}^{(2)}$	The quadratic sign algorithm (absorption)
Ξ_2	Predicate for comparing two quadratic surfaces

$\mathcal{B}^{(2)}$ The quadratic output-bit algorithm
$\mathsf{absorb}'^{(2)}$ The quadratic output-bit algorithm (absorption)
$\mathsf{same\_ratio}^{(2)}$ Singularity of $2 \times 2 \times 2$ tensors
$\mathcal{Q}$ The quadratic algorithm
$\mathsf{ssg}^{(n)}$ Sum of signs of $m_1, \ldots, m_{2^n}$
$\mathcal{S}^{(n)}$ The multilinear sign algorithm for n variables
$\mathsf{absorb}^{(n)}$ The multilinear sign algorithm for n variables (absorption)
Ξ_n Predicate for comparing two multilinear surfaces
$\mathcal{B}^{(n)}$ The multilinear output-bit algorithm for n variables
$\mathsf{absorb}'^{(n)}$ The multilinear output-bit algorithm for n variables (absorption)
$\mathsf{same\_ratio}^{(n)}$ Singularity of 2×2^n tensors
$\mathcal{M}^{(n)}$ The multilinear algorithm for n variables

References

[1] B.P. Bates, Self-matching and interleaving in some integer sequences and the Gauss map, PhD thesis, University of Wollongong, 2001.
[2] Y. Bertot, Simple canonical representation of rational numbers, in: H. Geuvers, F. Kamareddine (Eds.), Proceedings of the Workshop on Mathematics, Logic and Computation, Eindhoven, MLC'03, in: Electron. Notes Theor. Comput. Sci., vol. 85.7, Elsevier Science, Amsterdam, 2003.
[3] A. Brocot, Calcul des rouages par approximation, nouvelle méthode, Revue chronométrique. Journal des horlogers, scientifique et pratique 3 (1861) 186–194.
[4] J.H. Conway, On Numbers and Games, Academic Press, London, 1976.
[5] The Coq Development Team, The Coq proof assistant reference manual, Version 8.0, LogiCal Project, April 2004, http://coq.inria.fr/doc/main.html (cited 7 March 2006).
[6] A. Edalat, P.J. Potts, A new representation for exact real numbers, in: S. Brookes, M. Mislove (Eds.), Mathematical Foundations of Programming Semantics, Thirteenth Annual Conference (MFPS XIII), Carnegie Mellon University, Pittsburgh, PA, USA, March 23–26, 1997, in: Electron. Notes Theor. Comput. Sci., vol. 6, Elsevier Science, Amsterdam, 1997.
[7] R.W. Gosper, HAKMEM, Item 101 B, http://www.inwap.com/pdp10/hbaker/hakmem/cf.html#item101b (cited 7 March 2006), Feb. 29 1972, MIT AI Laboratory Memo No. 239.
[8] R.W. Gosper, Continued fraction arithmetic, Unpublished draft paper, text available at http://www.tweedledum.com/rwg/cfup.htm (cited 7 March 2006), 1978.
[9] R.E. Graham, D.E. Knuth, O. Patashnik, Concrete Mathematics. A Foundation for Computer Science, second ed., Addison-Wesley, Reading, MA, 1994.
[10] B. Hayes, On the teeth of wheels, American Scientist 88 (4) (July–August 2000) 296–300.
[11] J. Hughes, M. Niqui, Admissible digit sets and a modified Stern–Brocot representation, Technical Report NIII-R0401, Nijmegen Institute for Computer and Information Sciences, January 2004, http://www.cs.ru.nl/research/reports/full/NIII-R0401.pdf (cited 7 March 2006).
[12] M. Konečný, Many-valued real functions computable by finite transducers using IFS-representations, PhD thesis, School of Computer Science, The University of Birmingham, October 2000.
[13] P. Kornerup, D. Matula, An on-line arithmetic unit for bit-pipelined rational arithmetic, J. Parallel Distrib. Comput. 5 (1988) 310–330.
[14] P. Kornerup, D. Matula, LCF: A lexicographic binary representation of the rationals, J. Universal Comput. Sci. 1 (7) (July 1995) 484–503.
[15] P. Kornerup, D.W. Matula, An algorithm for redundant binary bit-pipelined rational arithmetic, IEEE Trans. Comput. C-39 (8) (August 1990) 1106–1115.
[16] D. Lester, Effective continued fractions, in: N. Burgess, L. Ciminiera (Eds.), 15th IEEE Symposium on Computer Arithmetic: ARITH-15 2001: Proceedings, Vail, CO, 11–13 June, 2001, IEEE Computer Society Press, 2001, pp. 163–172.
[17] P. Liardet, P. Stambul, Algebraic computations with continued fractions, J. Number Theory 73 (1998) 92–121.
[18] L.E. Mamane, Surreal numbers in Coq, Master's thesis, Technische Universiteit Eindhoven, July 2003, http://library.tue.nl/csp/dare/LinkToRepository.csp?recordnumber=568131 (cited 7 March 2006).
[19] V. Ménissier-Morain, Arithmétique exacte, conception, algorithmique et performances d'une implémentation informatique en précision arbitraire, Thèse, Université Paris 7, December 1994.
[20] M. Niqui, Formalising exact arithmetic: representations, algorithms and proofs, PhD thesis, Radboud Universiteit Nijmegen, September 2004.
[21] M. Niqui, Y. Bertot, http://coq.inria.fr/contribs/QArith-Stern-Brocot.html (cited 7 March 2006), May 2003. Files compatible with Coq V8.0.
[22] M. Niqui, Y. Bertot, QArith: Coq formalisation of lazy rational arithmetic, in: S. Berardi, M. Coppo, F. Damiani (Eds.), Types for Proofs and Programs: International Workshop, TYPES 2003, Torino, Italy, April 30–May 4, 2003, in: Lecture Notes in Comput. Sci., vol. 3085, Springer-Verlag, Berlin, 2004, pp. 309–323. Revised Selected Papers.
[23] P.J. Potts, Exact real arithmetic using Möbius transformations, PhD thesis, University of London, Imperial College, July 1998.
[24] P.J. Potts, A. Edalat, Exact real computer arithmetic, Technical Report DOC 97/9, Department of Computing, Imperial College, March 1997.
[25] G.N. Raney, On continued fractions and finite automata, Math. Ann. 206 (1973) 265–283.
[26] M.A. Stern, Ueber eine zahlentheoretische Funktion, Journal für die Reine und Angewandte Mathematik 55 (1858) 193–220.
[27] J.E. Vuillemin, Exact real computer arithmetic with continued fractions, IEEE Trans. Comput. 39 (8) (August 1990) 1087–1105.