Computing with Real Numbers

I. The LFT Approach to Real Number Computation

II. A Domain Framework for Computational Geometry

Abbas Edalat[1] and Reinhold Heckmann[2]

[1] Department of Computing, Imperial College, 180 Queens Gate,
London SW7 2BZ, UK.
e-mail: ae@doc.ic.ac.uk

[2] AbsInt Angewandte Informatik GmbH, Stuhlsatzenhausweg 69,
D-66123 Saarbrücken, Germany.
e-mail: heckmann@absint.com

Abstract. We introduce, in Part I, a number representation suitable for exact real number computation, consisting of an exponent and a mantissa, which is an infinite stream of signed digits, based on the interval $[-1, 1]$. Numerical operations are implemented in terms of *linear fractional transformations* (LFT's). We derive lower and upper bounds for the number of argument digits that are needed to obtain a desired number of result digits of a computation, which imply that the complexity of LFT application is that of multiplying n-bit integers. In Part II, we present an accessible account of a domain-theoretic approach to computational geometry and solid modelling which provides a data-type for designing robust geometric algorithms, illustrated here by the convex hull algorithm.

Part I: The LFT Aproach to Real Number Computation

1 Introduction

Computing with real numbers is one of the main applications of "computers". Yet real numbers are infinite mathematical objects (digit sequences, Cauchy sequences, nested sequences of intervals, or the like). Within finite time, one may only hope to obtain a finite part of a real number, giving an approximation to some accuracy.

This also means that comparison on real numbers is undecidable. Consider the predicate $x > 0$ applied to the number x represented as the nested sequence of intervals $([-\frac{1}{n}, \frac{1}{n}])_{n \geq 0}$. Within finite time, only a finite number of these intervals can be inspected, which always contain positive as well as negative numbers so that no decision on the sign of x is possible.

The problems mentioned above can be avoided by restricting attention to some subset of real numbers which can be finitely described. An obvious choice are the *rational numbers*, but this means that operations such as square root or tangent are not possible. A larger such set consists of the *algebraic numbers*, i.e., the roots of integer polynomials. With algebraic numbers, square roots and higher roots are possible, but trigonometric functions such as sine, cosine or tangent are still not supported.

Usually, a different approach is chosen. A finite set of machine-representable *floating-point numbers* is singled out, and fast operations are provided which approximate the standard operations and functions: if, say, the square root of a floating-point number is computed, then the resulting floating-point number is usually not the exact mathematical answer, but a number very close to it. The errors introduced by these approximations are known as *round-off errors*, and the easiest approach is to simply ignore them because they are so small. Yet in certain situations, round-off errors may accumulate to yield a big error. An example where this happens is the following number sequence defined by Jean-Michel Muller (found in [40]):

$$a_0 = \frac{11}{2}, \qquad a_1 = \frac{61}{11}, \qquad a_{n+1} = 111 - \frac{1130 - \frac{3000}{a_{n-1}}}{a_n}.$$

With the Unix program `bc`, one can compute with an arbitrary, but fixed number of decimal places. Let $a_n^{(k)}$ be the sequence element a_n computed with an accuracy of k decimal places. Computing with 5 decimal places yields the following results (rounded to 3 places for presentation):

$a_0^{(5)}$	5.500	$a_4^{(5)}$	5.648	$a_8^{(5)}$	103.738
$a_1^{(5)}$	5.545	$a_5^{(5)}$	5.242	$a_9^{(5)}$	100.209
$a_2^{(5)}$	5.590	$a_6^{(5)}$	−3.241	$a_{10}^{(5)}$	100.012
$a_3^{(5)}$	5.632	$a_7^{(5)}$	283.1	$a_{11}^{(5)}$	100.001

From this, one gets the impression that the sequence converges to 100. To confirm this impression, we compute the number a_{100} with an increasing number of decimal places:

$a_{100}^{(5)}$	100.0^41	$a_{100}^{(110)}$	$100.0^792\ldots$
$a_{100}^{(30)}$	$100.0^{29}1$	$a_{100}^{(120)}$	$-3.790\ \ldots$
$a_{100}^{(60)}$	$100.0^{57}997$	$a_{100}^{(130)}$	$5.9^78697\ldots$
$a_{100}^{(100)}$	$100.0^{17}98\ldots$	$a_{100}^{(140)}$	$5.9^787925\ldots$

Here, the "exponents" indicate the number of repetitions; for instance, 100.0^41 means 100.00001. As expected, the computations with 5, 30, and 60 decimal places show that a_{100} is close to the presumable limit 100. They are consistent in their result value, and it is tempting to think "I know that round-off errors may lead to wrong results, but if I increase the precision from 30 to 60 and the result obtained with 30 digits is confirmed, then it must be accurate." Yet the computations with 100 and 110 decimal places indicate that a_{100} is less close to 100 than expected, and worse, the computations with 120, 130, and 140 decimal places show that all the decimals obtained from the less precise computations were wrong. Or do the more precise computations yield wrong answers? What is the correct answer after all? Using the approach to exact real arithmetic presented in the sequel, one can verify that the number $a_{100}^{(140)}$ computed with 140 decimal places is an accurate approximation of the real value of a_{100} (and with a bit of mathematical reasoning, one can show that the sequence converges to 6, not to 100). Thus, on the positive side, we see that there is a precision (140) which yields the right answer for a_{100}, but in programs such as bc, one has to fix this precision in advance, and without a detailed analysis of the problem, it is unclear which precision will be sufficient (all precisions up to 110 give completely wrong, but consistent answers near 100).

In the approach to Exact Real Arithmetic presented here, one need not specify a fixed precision in advance. Instead, a real number is set up by some operations and functions, and then one may ask the system to evaluate this number up to a certain precision. The result will be an interval which approximates the real number with the required precision, and it is actually *guaranteed* that the number really is contained in this interval: with this arithmetic, it is impossible to get wrong answers (well, sometimes it may take very long to get an answer, but once the answer is there, it is trustworthy).

1.1 Overview

In Section 2, we introduce a number representation suitable for our purposes, consisting of an exponent and a mantissa, which is an infinite stream of signed digits. A few simple operations like $-x$ and $|x|$ are implemented directly on this representation. All other operations are implemented in terms of *linear fractional transformations* (LFT's). Individual LFT's act on number representations and digit streams in a uniform way which is fixed once and for all. Thus they provide a high-level framework for implementing functions without the need to think about their action on the low-level digit streams.

LFT's and basic LFT operations are introduced in Section 3. Section 4 studies monotonicity properties of general functions, in particular LFT's. Such properties are useful in the design and analysis of algorithms. In Section 5, we characterize those LFT's which map the *base interval* $[-1, 1]$ into itself (*refining LFT's*). The action of refining LFT's on digit streams is defined in Section 6: the absorption of argument digits into an LFT, and the emission of result digits from an LFT. Absorption and emission are the main ingredients of an algorithm that computes the result of applying an LFT to a real number (Section 6.3). Section 6.5 contains some example runs of this algorithm.

In Section 7, we derive lower and upper bounds for the number of argument digits that are needed to obtain a desired number of result digits of an LFT application. This information is complemented by information about the complexity of individual absorptions and emissions (Section 8). Taken together, these results imply that LFT application is quadratic in the number n of emitted digits—provided that digits are absorbed and emitted one by one. If many digits are absorbed and emitted at once, the complexity can be reduced to that of multiplying n-bit integers (Section 9).

All basic arithmetic operations are special instances of LFT application. In Section 10, the results about general LFT application are specialized to addition, multiplication, and reciprocal $1/x$. Transcendental functions can be implemented as infinite LFT expansions. Section 11 defines the semantics of such expansions, and shows how they can be derived from Taylor expansions (Section 11.4) or continued fraction expansions (Section 11.5). In Section 12, this knowledge is used to implement exponential function and natural logarithm; other functions are handled in the exercises.

Sections 13–18 present a domain-theoretic framework for computational geometry. Section 19 contains historical remarks to both parts, and Section 20 contains exercises.

2 Digit Streams

In the approach to real number computation presented here, (computable) real numbers are represented as potentially infinite streams of digits. At any time, a finite prefix of this stream has already been evaluated, e.g., $\pi = 3.14159\cdots$, and there is a method to compute larger finite prefixes on demand.

A finite prefix of the digit stream denotes an interval, namely the set of all real numbers whose digit streams start with this prefix. For instance, the prefix 3.14 denotes the interval $[3.14, 3.15]$ since all numbers starting with 3.14 are between $3.14000\cdots$ and $3.14999\cdots = 3.15$. The longer the prefix, the smaller the interval, e.g., 3.141 denotes $[3.141, 3.142]$. In this way, the digit stream denotes by means of its prefixes a nested sequence of intervals whose intersection contains exactly one number, namely the real number given by the digit stream.

The closed intervals of $\mathbb{R}$ form a *domain* when ordered under opposite inclusion. A nested sequence of intervals is an increasing chain in this domain, with its intersection as the least upper bound. The real numbers themselves are in one-to-one correspondence to the maximal elements of this domain, namely the degenerate intervals $[x, x]$. The Scott topology of the interval domain induces the usual topology on $\mathbb{R}$ via this embedding.

2.1 The Failure of Standard Number Systems

The examples above are based on the familiar *decimal system*, which is actually unsuitable for exact arithmetic (Brouwer [10]). We shall demonstrate this by means of an example, and note that similar examples exist for bases different from 10, i.e., this is a principal problem affecting all standard positional number systems.

Consider the task of computing the product $y = 3 \cdot x$ where x is given by the decimal representation $\xi = 0.333\cdots$. Mathematically, the result is given by the decimal representation $\eta = 0.999\cdots$, but is it possible to compute this result? Recall that at any time, only a finite prefix of ξ is known, and this finite prefix is the only source of information available to produce a finite prefix of the result η.

Assume we know the prefix 0.333 of ξ. Is this sufficient to determine the first digit of η? Unfortunately not, because the prefix 0.333 denotes the interval $[0.333, 0.334]$, which gives $[0.999, 1.002]$ when multiplied by 3. So we know that η should start with 0. or 1., but we do not yet know which is the right one, since neither the interval $[0, 1]$ denoted by 0. nor the interval $[1, 2]$ denoted by 1. covers the output interval $[0.999, 1.002]$. Worse, it is easy to see that this happens with all prefixes of the form $0.33\cdots3$. Hence if ξ is the stream $0.333\cdots$ with '3' forever, we can never output the first digit of η since no finite amount of information from ξ is sufficient to decide whether η should start with 0. or 1..

A solution to this problem is to admit negative digits ($-1, \ldots, -9$ in base 10). If we now find that ξ begins with 0.333, we may safely output '1.' (even 1.00) as a prefix of η since we can compensate by negative digits if it turns out later that the number represented by ξ is less than $1/3$, and so the result is actually smaller than 1. More formally, the interval denoted by the prefix 0.333 is now $[0.332, 0.334]$, since the smallest possible extension of 0.333 is no longer $0.33300\cdots$, but $0.333(-9)(-9)\cdots$. This interval yields $[0.996, 1.002]$ when multiplied by 3, which is contained in the interval $[0.99, 1.01]$ represented by the prefix 1.00, i.e., we can safely output 1.00 as the beginning of the output stream.

2.2 Signed Positional Systems

Signed positional systems are variants of standard positional systems which admit negative as well as positive digits. Like the standard systems, they are characterised by a base r, which is an integer $r \geq 2$. Once the base is fixed, the set of possible digits is taken as $D_r = \{d \in \mathbb{Z} \mid |d| < r\}$. For $r = 10$, we obtain $D_{10} = \{-9, -8, \ldots, 0, 1, \ldots, 9\}$ (*signed decimal system*), but the *signed binary system* with $r = 2$ and $D_2 = \{-1, 0, 1\}$ is practically more important. Most of these lecture notes deal with the case of base 2, which will therefore be the default case when the index r is omitted.

To avoid a special notation for the "decimal" (or "binary") point, let's assume it is always at the beginning of the digit stream. Then an (infinite) digit stream $\xi = \langle d_1, d_2, d_3, \cdots \rangle$ with $d_i \in D_r$ represents the real number $[\xi]_r = \sum_{i=1}^{\infty} d_i r^{-i}$ as usual. A finite digit sequence δ represents the set $[\delta]_r$ of all numbers $[\delta\xi]_r$ which are represented by extensions of δ to an infinite stream. For $\delta = \langle d_1, d_2, \cdots, d_n \rangle$, this set can be determined as the interval $[\delta]_r = [\sum_{i=1}^{n} d_i r^{-i} - r^{-n}, \sum_{i=1}^{n} d_i r^{-i} + r^{-n}]$ of length $2r^{-n}$. Note that the empty prefix $\langle\rangle$ ($n = 0$) denotes the interval $[-1, 1]$, which is the set of all real numbers representable by now. For the other ones, see Section 2.3 below.

In the sequel, we shall usually omit the parentheses and commas in digit sequences to obtain a more compact notation. Instead, we shall write concrete examples of digits and digit sequences in a special style, e.g., $\mathbf{4711}$ for $\langle 4,7,1,1\rangle$, to distinguish these sequences as syntactic objects from the numbers they denote. For further notational convenience, the minus sign becomes a bar within digit sequences, e.g., we write $\mathbf{1\bar{1}01}$ for the sequence $\langle 1,-1,0,1\rangle$. The digit sequence which results from attaching a single digit d to a sequence ξ will be written as $d:\xi$ (like "cons" in the lazy functional languages Haskell and Miranda). Unlike these languages, we shall abbreviate $d_1 : d_2 : \xi$ by $d_1 d_2 : \xi$.

What is then the proper semantic meaning of this "cons" operation? For infinite streams, we may calculate

$$[d:\xi]_r \;=\; d\cdot r^{-1} + \sum_{i=2}^{\infty} \xi_{i-1} r^{-i} \;=\; \frac{1}{r}\left(d + \sum_{i=1}^{\infty} \xi_i r^{-i}\right) \;=\; \frac{1}{r}(d+[\xi]_r)\,.$$

Hence, we have $[d:\xi]_r = A_d^r([\xi]_r)$ where A_d^r denotes the affine function with $A_d^r(x) = \frac{x+d}{r}$. A similar calculation can be done for finite digit sequences denoting intervals; the result is again $[d:\xi]_r = A_d^r([\xi]_r)$, but this time, both sides are intervals, and for an interval I, $A_d^r(I)$ is the image of I under A_d^r, which may as well be obtained as $A_d^r([u,v]) = [A_d^r(u), A_d^r(v)]$. For finite digit sequences, these considerations lead to an alternative characterisation of $[d_1\cdots d_n]_r$ as $A_{d_1}^r(\cdots A_{d_n}^r([-1,1])\cdots)$.

In contrast to the "cons" operation, the "tail" operation (omitting the first digit) has no semantic meaning. In base 2, $\mathbf{010}\ldots$ and $\mathbf{1\bar{1}0}\ldots$ both represent the number $\frac{1}{4}$, but their tails $\mathbf{10}\ldots$ and $\mathbf{\bar{1}0}\ldots$ represent two different numbers ($\frac{1}{2}$ and $-\frac{1}{2}$).

Let's now consider the practically important case $r=2$, $D=\{-1,0,1\}$ more closely. Here, we have (suppressing the index 2) $A_{\bar{1}}(x) = \frac{1}{2}(x-1)$, $A_0(x) = \frac{1}{2}x$, and $A_1(x) = \frac{1}{2}(x+1)$. All possible digit sequences up to length 2 and the intervals denoted by them are given by the following table:

$$\begin{array}{lll}
 & & [\mathbf{11}] = [\tfrac{1}{2},1] \\
 & [\mathbf{1}] = [0,1] & [\mathbf{10}] = [\tfrac{1}{4},\tfrac{3}{4}] \\
 & & [\mathbf{01}] = [\mathbf{1\bar{1}}] = [0,\tfrac{1}{2}] \\
[\,] = [-1,1] & [\mathbf{0}] = [-\tfrac{1}{2},\tfrac{1}{2}] & [\mathbf{00}] = [-\tfrac{1}{4},\tfrac{1}{4}] \\
 & & [\mathbf{0\bar{1}}] = [\mathbf{\bar{1}1}] = [-\tfrac{1}{2},0] \\
 & [\mathbf{\bar{1}}] = [-1,0] & [\mathbf{\bar{1}0}] = [-\tfrac{3}{4},-\tfrac{1}{4}] \\
 & & [\mathbf{\bar{1}\bar{1}}] = [-1,-\tfrac{1}{2}]
\end{array}$$

We see that the intervals overlap considerably, and some intervals are outright equal, e.g., $[\mathbf{1\bar{1}}]$ and $[\mathbf{01}]$. The latter observation can be strengthened to the fact that for all finite or infinite digit sequences δ, the sequences $\mathbf{1\bar{1}}:\delta$ and $\mathbf{01}:\delta$ are equivalent in the sense that they denote the same interval (finite case) or the same real number (infinite case). The semantic reason for this is $A_1 \circ A_{\bar{1}} = A_0 \circ A_1 = (x \mapsto \frac{1}{4}(x+1))$. Similarly, $\mathbf{\bar{1}1}:\delta$ and $\mathbf{0\bar{1}}:\delta$ are always equivalent.

Therefore, most real numbers have several (often infinitely many) different digit stream representations. This redundancy, or more precisely the overlapping which causes it is important for computability: if an output range crosses the border point 0 of $[\bar{\mathbf{1}}]$ and $[\mathbf{1}]$ and is sufficiently small, then it will be contained in $[\mathbf{0}]$, i.e., the digit $\mathbf{0}$ may be output. This observation may be strengthened as follows:

– If an interval $J \subseteq [-1, 1]$ has length $\ell(J) \leq \frac{1}{2}$, then it is contained in (at least) one of the three digit intervals $[\bar{\mathbf{1}}]$, $[\mathbf{0}]$, $[\bar{\mathbf{1}}]$.

An interval J with $\frac{1}{2} < \ell(J) \leq 1$ may or may not fit into one of the three digit intervals; consider $[-\epsilon, \frac{1}{2} + \epsilon]$ which does not fit for $\epsilon > 0$, and $[0, l]$ which fits into $[\mathbf{1}] = [0, 1]$ for $l \leq 1$. Finally, an interval J with $\ell(J) > 1$ cannot fit into any of the three digit intervals.

These observations can be generalised to digit sequences of length greater than 1 and arbitrary bases r as follows:

Proposition 2.1. *Let $J \subseteq [-1, 1]$ be an interval.*

1. *If $\ell(J) \leq r^{-n}$, then $J \subseteq [\delta]_r$ for some digit sequence δ of length n in base r.*
2. *If $J \subseteq [\delta]_r$ for some digit sequence δ of length n in base r, then $\ell(J) \leq 2r^{-n}$.*

2.3 Exponents

We have seen that a signed positional number system as defined above can only represent numbers x with $|x| \leq 1$ by digit streams. To obtain representations for real numbers x of any size, one may write x as $r^e \cdot x'$ where r^e is a power of the base and x' satisfies $|x'| \leq 1$ so that it can be represented by a digit stream. In principle, exponents $e \geq 0$ are sufficient, but allowing arbitrary $e \in \mathbb{Z}$ has its virtues. Thus, we arrive at representations $(e \parallel \xi)$ where e is an integer (called *exponent*) and ξ is a digit stream (called *mantissa*), and $(e \parallel \xi)$ represents $[(e \parallel \xi)]_r = r^e \cdot [\xi]_r$. Semantically, the attachment of the exponent can again be captured by an affine map, namely $[(e \parallel \xi)]_r = E^r_e([\xi]_r)$, where E^r_e is given by $E^r_e(x) = r^e \cdot x$.

The resulting number representation is similar to the familiar exponent-mantissa representation. The differences are that the mantissa is (potentially) infinite and may contain negative digits, and that no leading sign is required to represent negative numbers. (A further syntactic difference is that the exponent comes first; this reflects the fact that all algorithms deal with the exponent first before working with the mantissa.)

Clearly, the exponent in the number representation is not unique. Since $[\mathbf{0} : \xi]_r = \frac{1}{r}[\xi]_r$, a representation $(e \parallel \xi)$ can always be replaced by $(e+1 \parallel \mathbf{0} : \xi)$, or more generally by $(e + k \parallel \mathbf{0}^k : \xi)$, where $\mathbf{0}^k : \xi$ means that k $\mathbf{0}$-digits are attached to the beginning of ξ. On the other hand, we may remove leading $\mathbf{0}$-digits from ξ and reduce (*refine*) the exponent accordingly: $[(e \parallel \mathbf{0} : \xi)]_r = [(e-1 \parallel \xi)]_r$, or more generally $[(e \parallel \mathbf{0}^k : \xi)]_r = [(e - k \parallel \xi)]_r$.

Note that $\mathbf{0}$-digits may be squeezed out of a digit stream even if it does not begin with a $\mathbf{0}$-digit. For instance, in base $r = 2$, we have seen that $\mathbf{1}\bar{\mathbf{1}} : \xi$ and $\mathbf{01} : \xi$ are equivalent, and so are $\bar{\mathbf{1}}\mathbf{1} : \xi$ and $\mathbf{0}\bar{\mathbf{1}} : \xi$. Thus, we have $[(e \parallel \mathbf{1}\bar{\mathbf{1}} : \xi)] = [(e-1 \parallel \mathbf{1} : \xi)]$ and $[(e \parallel \bar{\mathbf{1}}\mathbf{1} : \xi)] = [(e-1 \parallel \bar{\mathbf{1}} : \xi)]$.

Refinement of the exponent is no longer possible iff the mantissa ξ starts with one of $\mathbf{10}$, $\mathbf{11}$, $\bar{\mathbf{1}}\mathbf{0}$, or $\bar{\mathbf{1}}\bar{\mathbf{1}}$. We call a representation with this property *normalised.* A normalised mantissa represents a number x with $\frac{1}{4} \leq |x| \leq 1$. All real numbers except 0 have normalised representations, but in contrast to the familiar case of unsigned digits, the exponents of two normalised representations for the same number may still differ by 1, e.g., $\frac{1}{3} = [(-1 \parallel (\mathbf{10})^\omega)] = [(0 \parallel \mathbf{1}(\mathbf{0}\bar{\mathbf{1}})^\omega)]$.

The computation of a real number y (more exactly, one of its representations) generally proceeds in the following stages:

1. Obtain an upper bound for the exponent of y.
2. Refine the exponent until it is sufficiently small or the representation is normalised.
3. Compute prefixes of the mantissa according to the required precision.

In simple cases, the exponent of the result is immediately known, but sometimes, considerable work is to be done in the first two stages.

2.4 Calculations with Digit Streams

Suppose we want to implement a function $f : \mathbb{R} \to \mathbb{R}$ which takes real numbers to real numbers. Then we need to find a corresponding function φ on *representations*, i.e., a function φ that maps representations $(e \parallel \xi)$ of x into representations $(e' \parallel \eta)$ of $f(x)$. Often, this function will be based on some function φ_0 that maps digit streams into digit streams. Algorithms for such stream functions can usually be specified recursively in the spirit of a lazy functional programming language such as Haskell or Miranda.

We are now ready to present the implementations of a few simple functions (and constants), always assuming base $r = 2$. We shall usually not distinguish between a stream ξ and its denotation $[\xi]$, nor between a function f and its representation φ.

Zero may be represented by $(0 \parallel \mathbf{0}^\omega)$, and *one* by $(0 \parallel \mathbf{1}^\omega)$ or $(1 \parallel \mathbf{10}^\omega)$.

Negation $-x$ can be implemented by leaving the exponent alone, and negating (the number represented by) the mantissa: $-(e \parallel \xi) \; = \; (e \parallel -\xi)$.
The latter can be done by flipping all digits around:

$$-(\mathbf{1} : \xi) \; = \; \bar{\mathbf{1}} : (-\xi), \qquad -(\mathbf{0} : \xi) \; = \; \mathbf{0} : (-\xi), \qquad -(\bar{\mathbf{1}} : \xi) \; = \; \mathbf{1} : (-\xi).$$

Absolute value $|x|$ can also be realised by acting on the mantissa:
$|(e \parallel \xi)| \; = \; (e \parallel |\xi|)$.
As long as the leading digit of ξ is $\mathbf{0}$, we do not know whether $[\xi]$ is positive or negative. But because of $[\mathbf{0} : \xi] = \frac{1}{2}[\xi]$ and $|\frac{1}{2}x| = \frac{1}{2}|x|$ we can safely output a $\mathbf{0}$-digit for every $\mathbf{0}$-digit we meet: $\quad |\mathbf{0} : \xi| \; = \; \mathbf{0} : |\xi|$.
Once the first non-zero digit has been found, we know $[\xi] \geq 0$ or $[\xi] \leq 0$, and can switch to the identity stream function or negation:

$$|\mathbf{1} : \xi| \; = \; \mathbf{1} : \xi, \qquad |\bar{\mathbf{1}} : \xi| \; = \; \mathbf{1} : (-\xi)\,.$$

Other operations. Implementations of the minimum function $\min(x,y)$ and addition $x+y$ in this framework are straightforward (see also Exercise 1). Multiplication is a bit more difficult, but division already requires some ingenuity, and there is no immediate way to obtain functions like square root, exponential, logarithm, etc. Fortunately, *linear fractional transformations* (LFT's) provide a high-level framework that makes the implementation of such real number operations much easier. Individual LFT's act on number representations and digit streams in a uniform way which is fixed once and for all. The desired real number operations may then be implemented in terms of LFT expressions, without the need to think about their action on the low-level digit streams. (Another approach was used by Plume [42] who worked on digit streams using auxiliary representations and an auxiliary limit function. These also provide an abstraction from the underlying digit streams.)

3 Linear Fractional Transformations (LFT's)

We have already seen that the semantic meaning of digits and exponents can be captured by certain *affine transformations*: $[d:\xi]_r = A_d^r([\xi]_r)$ with $A_d^r(x) = \frac{x+d}{r}$, and $[(e \parallel \xi)]_r = E_e^r([\xi]_r)$ with $E_e^r(x) = r^e \cdot x$. The general form of these affine transformations is $A(x) = ax+b$ with two fixed parameters a and b. Considering affine transformations would already be sufficient to obtain some useful results, but to handle division and certain transcendental functions, one needs the more general linear fractional transformations or LFT's.

3.1 One-Dimensional LFT's (1-LFT's) and Matrices

A *one-dimensional linear fractional transformation* (1-LFT), also called *Möbius transformation*, is a function of the form $L(x) = \frac{ax+c}{bx+d}$ with four fixed parameters a, b, c, and d. In general, these parameters are arbitrary real (or even complex) numbers, but we shall usually only consider 1-LFT's with integer parameters.

The notion of 1-LFT includes that of affine transformation. A 1-LFT $\frac{ax+c}{bx+d}$ is *affine* if and only if $b=0$; in this case it becomes $\frac{a}{d}x + \frac{c}{d}$.

For ease of notation, we abbreviate the function $x \mapsto \frac{ax+c}{bx+d}$ by $\left\langle\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right\rangle$. The following are some examples of 1-LFT's:

$x \mapsto x$	$\left\langle\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right\rangle$	$x \mapsto -x$	$\left\langle\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\right\rangle$
$x \mapsto x+1$	$\left\langle\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right\rangle$	$x \mapsto 3x$	$\left\langle\begin{smallmatrix} 3 & 0 \\ 0 & 1 \end{smallmatrix}\right\rangle$
$x \mapsto \frac{1}{x}$	$\left\langle\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right\rangle$	$x \mapsto \frac{2x+3}{4x+5}$	$\left\langle\begin{smallmatrix} 2 & 3 \\ 4 & 5 \end{smallmatrix}\right\rangle$
$x \mapsto A_d^r(x) = \frac{x+d}{r}$	$\left\langle\begin{smallmatrix} 1 & d \\ 0 & r \end{smallmatrix}\right\rangle$	$x \mapsto E_e^r(x) = r^e \cdot x$	$\left\langle\begin{smallmatrix} r^e & 0 \\ 0 & 1 \end{smallmatrix}\right\rangle$

The notation $\left\langle\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right\rangle$ for 1-LFT's looks similar to a 2-2-matrix $M = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$. Indeed, any such matrix $M = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ defines a 1-LFT, namely $\langle M \rangle = \left\langle\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right\rangle$, with

$\langle M\rangle(x) = \frac{ax+c}{bx+d}$. Yet this correspondence is not one-to-one: in a 1-LFT, common factors of the four parameters do not matter; $\left\langle\begin{smallmatrix} a & c \\ b & d\end{smallmatrix}\right\rangle$ and $\left\langle\begin{smallmatrix} ka & kc \\ kb & kd\end{smallmatrix}\right\rangle$ are the same 1-LFT if k is a non-zero number. Thus, we have $\langle M\rangle = \langle kM\rangle$ for $k \neq 0$. In fact, the opposite direction also holds: if $\langle M_1\rangle = \langle M_2\rangle$, then M_1 and M_2 differ only by a non-zero multiplicative factor. In particular, we have $\langle M\rangle = \langle -M\rangle$. As a slight normalisation, we usually present 1-LFT's in a way such that the lower right entry is non-negative ($d \geq 0$).

The matrix-like notation for 1-LFT's carries mathematical meaning because of the following:

Proposition 3.1. *The composition of two 1-LFT's L_1 and L_2 is again a 1-LFT. Composition of 1-LFT's corresponds to matrix multiplication: $\langle M_1\rangle \circ \langle M_2\rangle = \langle M_1 \cdot M_2\rangle$ (Exercise 2).*

Recall from linear algebra how two matrices are multiplied:

$$\begin{pmatrix} a & c \\ b & d\end{pmatrix} \cdot \begin{pmatrix} a' & c' \\ b' & d'\end{pmatrix} = \begin{pmatrix} aa'+cb' & ac'+cd' \\ ba'+db' & bc'+dd'\end{pmatrix} \tag{1}$$

If $b = b' = 0$, then also $ba'+db' = 0$, hence affinity is preserved by multiplication. The neutral element of matrix multiplication is the *identity matrix* $\mathrm{E} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1\end{smallmatrix}\right)$, whose 1-LFT $\left\langle\begin{smallmatrix} 1 & 0 \\ 0 & 1\end{smallmatrix}\right\rangle$ is the identity function. Recall further the important notion of the *determinant* of a matrix

$$\det \begin{pmatrix} a & c \\ b & d\end{pmatrix} = ad - bc \tag{2}$$

and its basic properties:

$$\det \mathrm{E} = 1 \qquad \det(A \cdot B) = \det A \cdot \det B \qquad \det(kM) = k^2 \cdot \det M \tag{3}$$

Because of the last equation above, the determinant is not a well-defined property of a 1-LFT (remember that $\langle kM\rangle = \langle M\rangle$ for $k \neq 0$). Yet the *sign* of the determinant is a perfect 1-LFT property because for $k \neq 0$, $\det(kM) \gtreqless 0$ iff $\det M \gtreqless 0$.

A matrix M is *non-singular* iff $\det M \neq 0$. The *inverse* of a non-singular matrix $M = \left(\begin{smallmatrix} a & c \\ b & d\end{smallmatrix}\right)$ is given by $\left(\begin{smallmatrix} a & c \\ b & d\end{smallmatrix}\right)^{-1} = \frac{1}{\det M}\left(\begin{smallmatrix} d & -c \\ -b & a\end{smallmatrix}\right)$. For 1-LFT's, the factor $\frac{1}{\det M}$ does not matter, and we may define the *pseudo inverse* M^* instead:

$$\begin{pmatrix} a & c \\ b & d\end{pmatrix}^* = \begin{pmatrix} d & -c \\ -b & a\end{pmatrix} \tag{4}$$

Note that the pseudo inverse of an integer matrix is again an integer matrix, and affinity ($b = 0$) is preserved as well. The following are the main properties of this notion (in the matrix world):

$$\begin{aligned} \mathrm{E}^* &= \mathrm{E} & (M^*)^* &= M \\ (k \cdot M)^* &= k \cdot M^* & (A \cdot B)^* &= B^* \cdot A^* \\ \det M^* &= \det M & M \cdot M^* = M^* \cdot M &= \det M \cdot \mathrm{E} \end{aligned} \tag{5}$$

Since *non-zero* factors do not matter for 1-LFT's, the last property gives the 1-LFT equation $\langle M^* \rangle \circ \langle M \rangle = \langle M \rangle \circ \langle M^* \rangle = \mathrm{id}$ for $\det M \neq 0$, i.e., $\langle M^* \rangle$ is the inverse function of $\langle M \rangle$.

3.2 Two-Dimensional LFT's (2-LFT's) and Tensors

The 1-LFT's defined above are functions of one argument, and as such, not suitable to capture the standard binary operations of addition $x+y$, subtraction $x-y$, multiplication $x \cdot y$, and division x/y. For this purpose, we introduce LFT's of two arguments (two-dimensional LFT's, shortly 2-LFT's).

A *two-dimensional linear fractional transformation* (2-LFT) is a function of the form $L(x,y) = \frac{axy+cx+ey+g}{bxy+dx+fy+h}$ with eight fixed parameters a, b, c, d, e, f, g, and h. For ease of notation, we write this function as $\left\langle \begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix} \right\rangle$. The following are some examples of 2-LFT's:

$(x,y) \mapsto x+y$	$\left\langle \begin{smallmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{smallmatrix} \right\rangle$	$(x,y) \mapsto x-y$	$\left\langle \begin{smallmatrix} 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{smallmatrix} \right\rangle$
$(x,y) \mapsto x \cdot y$	$\left\langle \begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{smallmatrix} \right\rangle$	$(x,y) \mapsto x/y$	$\left\langle \begin{smallmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{smallmatrix} \right\rangle$
$(x,y) \mapsto \frac{x+y}{1-xy}$	$\left\langle \begin{smallmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{smallmatrix} \right\rangle$	$(x,y) \mapsto \frac{2x+3}{4y+5}$	$\left\langle \begin{smallmatrix} 0 & 2 & 0 & 3 \\ 0 & 0 & 4 & 5 \end{smallmatrix} \right\rangle$

The notation $\left\langle \begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix} \right\rangle$ for 2-LFT's looks similar to a 2-4-matrix $T = \left(\begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix} \right)$, called *tensor*. The relation between tensors and 2-LFT's is similar to the relation between matrices and 1-LFT's. Any tensor T defines a 2-LFT $\langle T \rangle$. Two tensors define the same 2-LFT if and only if their entries differ by a non-zero multiplicative factor. Thus, $\langle T \rangle = \langle kT \rangle$ for $k \neq 0$; in particular $\langle T \rangle = \langle -T \rangle$. We usually present 2-LFT's in a way such that the lower right entry is non-negative ($h \geq 0$).

If the second argument of a 2-LFT $F = \left\langle \begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix} \right\rangle$ is a fixed number y, then $F|^y$ is a function in one argument, given by

$$F|^y(x) = F(x,y) = \frac{(ay+c)x + (ey+g)}{(by+d)x + (fy+h)} = \left\langle \begin{matrix} ay+c & ey+g \\ by+d & fy+h \end{matrix} \right\rangle (x)\,.$$

A similar calculation can be done if the first argument is a fixed number x, leading to another 1-LFT $F|_x$. Thus, if we define for tensors $T = \left(\begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix} \right)$

$$T|_x = \begin{pmatrix} ax+e & cx+g \\ bx+f & dx+h \end{pmatrix} \quad \text{and} \quad T|^y = \begin{pmatrix} ay+c & ey+g \\ by+d & fy+h \end{pmatrix} \tag{6}$$

then $\langle T|_x \rangle(y) = T(x,y)$ and $\langle T|^y \rangle(x) = T(x,y)$.

While there is no obvious way to compose two 2-LFT's in the framework presented here, there are several ways to compose a 2-LFT and a 1-LFT (or to multiply a tensor and a matrix). Let for the following M be a matrix and T a tensor.

First, the function F defined by $F(x,y) = \langle M\rangle(\langle T\rangle(x,y))$ is again a 2-LFT, namely $F = \langle MT\rangle$, where MT is an instance of ordinary matrix multiplication:

$$\begin{pmatrix} a' & c' \\ b' & d' \end{pmatrix} \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} = \begin{pmatrix} a'a+c'b & a'c+c'd & a'e+c'f & a'g+c'h \\ b'a+d'b & b'c+d'd & b'e+d'f & b'g+d'h \end{pmatrix} \quad (7)$$

Second, the function G defined by $G(x,y) = \langle T\rangle(\langle M\rangle(x), y)$ is again a 2-LFT, namely $G = \langle T \circledL M\rangle$, where $T \circledL M$ is a special purpose operation defined by

$$\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \circledL \begin{pmatrix} a' & c' \\ b' & d' \end{pmatrix} = \begin{pmatrix} aa'+eb' & ca'+gb' & ac'+ed' & cc'+gd' \\ ba'+fb' & da'+hb' & bc'+fd' & dc'+hd' \end{pmatrix} \quad (8)$$

Third, the function H defined by $H(x,y) = \langle T\rangle(x, \langle M\rangle(y))$ is again a 2-LFT, namely $H = \langle T \circledR M\rangle$, where $T \circledR M$ is a special purpose operation defined by

$$\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \circledR \begin{pmatrix} a' & c' \\ b' & d' \end{pmatrix} = \begin{pmatrix} aa'+cb' & ac'+cd' & ea'+gb' & ec'+gd' \\ ba'+db' & bc'+dd' & fa'+hb' & fc'+hd' \end{pmatrix} \quad (9)$$

All these operations are connected by various algebraic laws:

$$(M_1 \cdot M_2) \cdot T = M_1 \cdot (M_2 \cdot T) \qquad (T \circledL M_1) \circledR M_2 = (T \circledR M_2) \circledL M_1 \quad (10)$$

$$(M_1 \cdot T) \circledL M_2 = M_1 \cdot (T \circledL M_2) \qquad (M_1 \cdot T) \circledR M_2 = M_1 \cdot (T \circledR M_2) \quad (11)$$

$$(T \circledL M_1) \circledL M_2 = T \circledL (M_1 \cdot M_2) \qquad (T \circledR M_1) \circledR M_2 = T \circledR (M_1 \cdot M_2) \quad (12)$$

3.3 Zero-Dimensional LFT's (0-LFT's) and Vectors

In analogy to 1-LFT's which take one argument and 2-LFT's which take two arguments, there are also 0-LFT's $\left\langle \begin{smallmatrix} a \\ b \end{smallmatrix} \right\rangle$ which take no argument at all, but deliver the constant $\frac{a}{b}$.

The notation $\left\langle \begin{smallmatrix} a \\ b \end{smallmatrix} \right\rangle$ for 0-LFT's looks similar to a vector $\left(\begin{smallmatrix} a \\ b \end{smallmatrix} \right)$. Clearly, two vectors correspond to the same 0-LFT if and only if they differ by a non-zero multiplicative factor.

A 1-LFT $\left\langle \begin{smallmatrix} a & c \\ b & d \end{smallmatrix} \right\rangle$ can be applied to a 0-LFT $\left\langle \begin{smallmatrix} u \\ v \end{smallmatrix} \right\rangle$ resulting in a new 0-LFT $\left\langle \begin{smallmatrix} au+cv \\ bu+dv \end{smallmatrix} \right\rangle$. If the first argument of a 2-LFT $F = \left\langle \begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix} \right\rangle$ is a fixed 0-LFT $w = \left\langle \begin{smallmatrix} u \\ v \end{smallmatrix} \right\rangle$, then $F|_w$ is the 1-LFT $\left\langle \begin{smallmatrix} au+ev & cu+gv \\ bu+fv & du+hv \end{smallmatrix} \right\rangle$. Similarly, $F|^w = \left\langle \begin{smallmatrix} au+cv & eu+gv \\ bu+dv & fu+hv \end{smallmatrix} \right\rangle$.

These absorption rules can be used to deal with rational numbers in the real arithmetic. An expression like $\frac{1}{3}\pi$ can be set up as $\left\langle \begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{smallmatrix} \right\rangle (\left\langle \begin{smallmatrix} 1 \\ 3 \end{smallmatrix} \right\rangle, \pi)$ using the tensor for multiplication, and then simplified to $\left\langle \begin{smallmatrix} 1 & 0 \\ 0 & 3 \end{smallmatrix} \right\rangle (\pi)$. If only rational operations on rational numbers are performed, this is equivalent to a rational arithmetic, with the disadvantage that in general, denominators double in their bit size in every addition and multiplication. Alternatively, a rational number can be treated like any real number and transformed into a digit stream.

4 Monotonicity

By interval, we always mean a closed interval $[u, v]$ with $u \leq v$ in $\mathbb{R}$. If I is an interval and $f : I \to \mathbb{R}$ a continuous function, then its image $f(I)$ is again an interval. To actually determine the end points of $f(I)$, it is useful to know about the monotonicity of f.

A function $f : I \to \mathbb{R}$ is

- *increasing* if $x \leq y$ in I implies $f(x) \leq f(y)$,
- *decreasing* if $x \leq y$ in I implies $f(x) \geq f(y)$,
- *strictly increasing* if $x < y$ in I implies $f(x) < f(y)$,
- *strictly decreasing* if $x < y$ in I implies $f(x) > f(y)$,
- *monotonic* if it is increasing (on the whole of I) or decreasing (on the whole of I).

For monotonic functions, we also speak of their *monotonicity type*, which is $\uparrow$ for increasing functions, and $\downarrow$ for decreasing functions. Clearly, $f([u, v]) = [f(u), f(v)]$ for increasing f, and $f([u, v]) = [f(v), f(u)]$ for decreasing f. Hence for monotonic f, $f([u, v])$ is the interval spanned by the two values $f(u)$ and $f(v)$, extending from their minimum to their maximum. If J is another interval, then $f([u, v]) \subseteq J$ if and only if both $f(u)$ and $f(v)$ are in J.

Let $\langle M \rangle$ be a 1-LFT such that the denominator $bx + d$ of $\langle M \rangle(x) = \frac{ax+c}{bx+d}$ is non-zero for all x in an interval I. We call such a 1-LFT *bounded* on I since it avoids the value ∞ which formally occurs as a fraction with denominator 0. Analogous notions can be introduced for 2-LFT's.

A 1-LFT $f = \langle M \rangle$ which is bounded on I is a continuous function $f : I \to \mathbb{R}$, given by $f(x) = \frac{ax+c}{bx+d}$. Clearly, this function is differentiable with derivative $f'(x) = \frac{ad-bc}{(bx+d)^2}$. In this fraction, the denominator is always greater than 0 (it cannot be 0 since f was supposed to be bounded on I), while the numerator is a constant, namely $\det M$. Thus, the monotonicity behaviour of $\langle M \rangle$ depends only on the sign of $\det M$ (which is a meaningful notion for a 1-LFT):

- If $\det M > 0$, then $\langle M \rangle'(x) > 0$ for all x in I, hence $\langle M \rangle$ is strictly increasing.
- If $\det M < 0$, then $\langle M \rangle'(x) < 0$ for all x in I, hence $\langle M \rangle$ is strictly decreasing.
- If $\det M = 0$, then $\langle M \rangle'(x) = 0$ for all x in I, hence $\langle M \rangle$ is constant on I.

In any case, $\langle M \rangle$ is monotonic, and therefore, the remarks on monotonic functions given above apply. All this relies on the fact that we let the 1-LFT act on an interval; for instance, $\left\langle \begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix} \right\rangle = (x \mapsto \frac{1}{x})$ with $\det \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right) = -1$ is decreasing on $[1, 2]$ and on $[-2, -1]$, but not on $[-1, 1] \setminus \{0\}$.

We now turn to functions of two arguments. Let I and J be two intervals. Geometrically, their product set $I \times J$ is a rectangle. For a function $F : I \times J \to \mathbb{R}$, we define $F|_x : J \to \mathbb{R}$ for fixed x in I by $F|_x(y) = F(x, y)$, and dually $F|^y : I \to \mathbb{R}$ for fixed y in J by $F|^y(x) = F(x, y)$; these functions are the *sections* of F.

A function $F : I \times J \to \mathbb{R}$ is *monotonic* if all its sections $F|_x$ for $x \in I$ and $F|^y$ for $y \in J$ are monotonic. Recall that all the sections of a 2-LFT are 1-LFT's, and therefore monotonic by the results above. Hence, every 2-LFT is monotonic on every rectangle where it is bounded (i.e., its denominator avoids 0).

Proposition 4.1. *If $F : [u_1, u_2] \times [v_1, v_2] \to \mathbb{R}$ is continuous and monotonic, then its image $F([u_1, u_2] \times [v_1, v_2])$ is the interval spanned by the four corner values $F(u_1, v_1)$, $F(u_1, v_2)$, $F(u_2, v_1)$, and $F(u_2, v_2)$, i.e., it extends from the smallest of these values to the largest.*

Corollary 4.2. *If $F : [u_1, u_2] \times [v_1, v_2] \to \mathbb{R}$ is continuous and monotonic, then for all intervals J, the inclusion $F([u_1, u_2] \times [v_1, v_2]) \subseteq J$ holds if and only if all the corner values $F(u_1, v_1)$, $F(u_1, v_2)$, $F(u_2, v_1)$, and $F(u_2, v_2)$ are in J.*

If $F : I \times J \to \mathbb{R}$ is monotonic, then it may happen that some of the sections $F|^y$ are increasing, while some other sections $F|^y$ are decreasing. We say F is *increasing in the first argument* if all sections $F|^y$ for $y \in J$ are increasing. The properties to be decreasing in the first (or second) argument are defined analogously. We say F *has type* $(\uparrow, \downarrow)$ if F is increasing in the first argument and decreasing in the second. The 3 other types $(\uparrow, \uparrow)$, $(\downarrow, \uparrow)$, and $(\downarrow, \downarrow)$ are defined similarly.

Let's consider some examples. On $I_0 \times I_0 = [-1, 1]^2$, addition $F(x, y) = x + y$ has type $(\uparrow, \uparrow)$, subtraction $F(x, y) = x - y$ has type $(\uparrow, \downarrow)$, while multiplication $F(x, y) = x \cdot y$ is of course monotonic like all other 2-LFT's, but does not have any of the four types. For, $F|^1(x) = x$ is increasing, but $F|^{-1}(x) = -x$ is decreasing.

5 Bounded and Refining LFT's

Later, we shall apply LFT's to arguments given by digit streams. Of course, this makes only sense if the LFT is well-defined for arguments from the *"base interval"* $I_0 = [-1, 1]$, i.e., is *bounded* in the sense that its denominator avoids 0 for arguments from I_0. If we want the result to be represented by a digit stream as well, then the LFT should moreover be *refining*, i.e., map I_0 into itself.

In this section, we shall derive some criteria for LFT's to be bounded and refining, and prove some properties of these notions. These proofs involve some manipulations of absolute values, so that it is worthwhile to establish some properties of absolute values in the beginning. Recall

$$|x| \;=\; \max(x, -x) \qquad\qquad -|x| \;=\; \min(x, -x) \tag{13}$$

for real numbers x. The following lemma will be useful in dealing with sums.

Lemma 5.1.
$\max(|x + y|, |x - y|) \;=\; |x| + |y| \quad$ *and* $\quad |x + y| + |x - y| \;=\; 2 \max(|x|, |y|)$.

5.1 Bounded 1-LFT's

A 1-LFT $\left\langle\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right\rangle$ is *bounded* iff the denominator $D(x) = bx + d$ is non-zero for all $x \in I_0$. Since I_0 is an interval and D is continuous, this means either $D(x) > 0$

for all x in I_0, or $D(x) < 0$ for all x in I_0. Under the general assumption $d \geq 0$, the second case is ruled out because $D(0) = d$. To check $D(x) > 0$ for all $x \in I_0$, it suffices to consider the minimal value of D on $[-1, 1]$. For $b \geq 0$, this is $D(-1) = d - b$, and for $b \leq 0$, it is $D(1) = d + b$. In any case, the minimum is $d - |b|$. Therefore, we obtain:

Proposition 5.2. *$\left\langle\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right\rangle$ with $d \geq 0$ is bounded if and only if $d > |b|$. In this case, the denominator $bx + d$ is positive for all x in I_0.*

5.2 Bounded 2-LFT's

For a 2-LFT $F = \left\langle\begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix}\right\rangle$, the denominator is $D(x, y) = bxy + dx + fy + h$. We say F is bounded if D avoids 0 for (x, y) in I_0^2. Under the general assumption $h = D(0, 0) \geq 0$, this is again equivalent to positivity of D on I_0^2. Function D is monotonic; this is most easily seen by noting that $D = \left\langle\begin{smallmatrix} b & d & f & h \\ 0 & 0 & 0 & 1 \end{smallmatrix}\right\rangle$ is a 2-LFT. Hence, the range of possible values of D on I_0^2 is spanned by the four corner values $D(\pm 1, \pm 1)$. Thus, F is bounded iff the four values $b + d + f + h$, $-b - d + f + h$, $-b + d - f + h$, and $b - d - f + h$ are positive. Equivalently, this means

$$h > \max(b + d - f, b - d + f, -b + d + f, -b - d - f)\,. \tag{14}$$

In case of $b = 0$, the condition can be simplified to $h > |d| + |f|$ with the help of Lemma 5.1.

Proposition 5.3. *If $\left\langle\begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix}\right\rangle$ with $h \geq 0$ is bounded, then $h > \max(|b|, |d|, |f|)$.*

Proof. We start with (14). Adding the two relations $h > b+d-f$ and $h > b-d+f$ gives $2h > 2b$, and adding $h > -b + d + f$ and $h > -b - d - f$ yields $2h > -2b$. Together, $h > |b|$ follows. In a similar way, $h > |d|$ and $h > |f|$ can be derived. □

5.3 Refining 1-LFT's

A bounded 1-LFT $f = \left\langle\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right\rangle$ is *refining* if $f(I_0) \subseteq I_0$. Since f is monotonic, this is equivalent to the two conditions $f(-1) \in I_0$ and $f(1) \in I_0$, or $|f(-1)| \leq 1$ and $|f(1)| \leq 1$. With the assumption $d \geq 0$, the denominator of $f(x) = \frac{ax+c}{bx+d}$ is positive. Hence, the two conditions can be reformulated as $|c - a| \leq d - b$ and $|c + a| \leq d + b$, or $d \geq \max(|c - a| + b, |c + a| - b) = \max(c + a - b, c - a + b, -c + a + b, -c - a - b)$.

Note the similarity of this condition to the condition for a 2-LFT to be bounded (14); the only difference lies in the variable names and the relation symbol. Hence everything what has been said about bounded 2-LFT's holds here as well in an analogous way:

Proposition 5.4.
An affine 1-LFT $\left\langle\begin{smallmatrix} a & c \\ 0 & d \end{smallmatrix}\right\rangle$ with $d > 0$ is refining if and only if $d \geq |a| + |c|$.

Proposition 5.5. *If $\left\langle\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right\rangle$ with $d \geq 0$ is refining, then $d \geq \max(|a|, |b|, |c|)$.*

5.4 Refining 2-LFT's

A bounded 2-LFT $F = \left\langle \begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix} \right\rangle$ is *refining* if $F(I_0^2) \subseteq I_0$. Since F is monotonic, this is equivalent to the condition that all four corner values $F(\pm 1, \pm 1)$ are in I_0, or $|F(\pm 1, \pm 1)| \leq 1$. With the assumption $h \geq 0$, all denominators are positive. Hence, the four conditions can be reformulated as

$$\begin{array}{rclcrcl} |a+c+e+g| & \leq & b+d+f+h & \quad & |-a-c+e+g| & \leq & -b-d+f+h \\ |a-c-e+g| & \leq & b-d-f+h & \quad & |-a+c-e+g| & \leq & -b+d-f+h \end{array} \tag{15}$$

We now show that the lower right entry h of a refining 2-LFT dominates all other ones (under the assumption $h \geq 0$). First, we know from Prop. 5.3 that $h > |b|, |d|, |f|$. Adding the two equations in the first column of (15) gives $\max(|a+g|, |c+e|) \leq h+b$ with the help of Lemma 5.1. Similarly, adding the second column yields $\max(|a-g|, |c-e|) \leq h-b$. Next, adding $|a+g| \leq h+b$ and $|a-g| \leq h-b$ gives $\max(|a|, |g|) \leq h$, and adding the other two relations yields $\max(|c|, |e|) \leq h$.

Proposition 5.6. *If* $\left\langle \begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix} \right\rangle$ *with* $h \geq 0$ *is refining, then* $h \geq |a|, |c|, |e|, |g|$ *and* $h > |b|, |d|, |f|$.

6 LFT's and Digit Streams

Now we consider the application of (refining) LFT's to arguments from I_0. The LFT's will be represented by matrices, and the arguments and results by digit streams (exponents are handled later). We take the freedom to occasionally identify LFT's and their representing matrices, and thus to apply the LFT notions bounded, refining, monotonic etc. to the representing matrices as well.

6.1 Absorption of Argument Digits

Absorption into Matrices. Let $f = \langle M \rangle$ be a 1-LFT to be applied to a digit stream. Remember that a digit k in base r corresponds to an affine transformation A_k^r with $A_k^r(x) = \frac{x+k}{r}$. This is a special case of a 1-LFT, with matrix $A_k^r = \left(\begin{smallmatrix} 1 & k \\ 0 & r \end{smallmatrix} \right)$. Using this matrix, we may calculate

$$\langle M \rangle([k:\xi]_r) \;=\; \langle M \rangle(\langle A_k^r \rangle([\xi]_r)) \;=\; \langle M \cdot A_k^r \rangle([\xi]_r).$$

Thus, we may *absorb* the first digit of the argument stream into the matrix M by multiplying M with A_k^r from the right:

– Absorption: $M(k:\xi) \;=\; (M \cdot A_k^r)\,(\xi)$.

An explicit formula for the product $M \cdot A_k^r$ may be obtained by specialising Equation (1):

$$M \cdot A_k^r \;=\; \begin{pmatrix} a & c \\ b & d \end{pmatrix} \cdot \begin{pmatrix} 1 & k \\ 0 & r \end{pmatrix} \;=\; \begin{pmatrix} a & rc+ka \\ b & rd+kb \end{pmatrix} \tag{16}$$

For the following, let $M = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ and $M' = M \cdot A_k^r = \left(\begin{smallmatrix} a' & c' \\ b' & d' \end{smallmatrix}\right)$, where the actual values of a' etc. are given by (16).

1. If M is bounded with positive denominator, then so is M'.
   Proof: Let $D(x) = bx + d$ be the denominator of M, and $D'(x) = b'x + d'$ the denominator of M'. Both D and D' are 1-LFT's, namely $D = \left(\begin{smallmatrix} b & d \\ 0 & 1 \end{smallmatrix}\right)$ and $D' = \left(\begin{smallmatrix} b & rd+kb \\ 0 & 1 \end{smallmatrix}\right)$. By (16), $D \cdot A_k^r$ is $D'' = \left(\begin{smallmatrix} b & rd+kb \\ 0 & r \end{smallmatrix}\right)$. By hypothesis, $D(x) > 0$ for all x in I_0. Hence, $D(x) > 0$ for all $x \in A_k^r(I_0) \subseteq I_0$, and therefore, $D''(x) = D(A_k^r(x)) > 0$ for all x in I_0. From this, positivity of $D'(x) = r \cdot D''(x)$ immediately follows.
2. If M is refining, then so is M'.
   Proof: If $M(I_0) \subseteq I_0$, then $M'(I_0) = M(A_k^r(I_0)) \subseteq M(I_0) \subseteq I_0$.
3. If M is increasing (decreasing), then so is M'.
   Proof: M' is M composed with the increasing function A_k^r.

Absorption into Tensors. The absorption of a digit into a tensor T rests on a similar semantic foundation. It comes in two versions, depending on whether the digit is taken from the left or the right argument.

– Left absorption: $T\,(k : \xi,\ \eta) \ = \ (T \circledL A_k^r)\,(\xi,\ \eta)$.
– Right absorption: $T\,(\xi,\ k : \eta) \ = \ (T \circledR A_k^r)\,(\xi,\ \eta)$.

Explicit formulae for the products $T \circledL A_k^r$ and $T \circledR A_k^r$ may be obtained by specialising (8) and (9):

$$T \circledL A_k^r \ = \ \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \circledL \begin{pmatrix} 1 & k \\ 0 & r \end{pmatrix} \ = \ \begin{pmatrix} a & c & re+ka & rg+kc \\ b & d & rf+kb & rh+kd \end{pmatrix} \quad (17)$$

$$T \circledR A_k^r \ = \ \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \circledR \begin{pmatrix} 1 & k \\ 0 & r \end{pmatrix} \ = \ \begin{pmatrix} a & rc+ka & e & rg+ke \\ b & rd+kb & f & rh+kf \end{pmatrix} \quad (18)$$

For the following, let $T' = T \circledL A_k^r$ or $T' = T \circledR A_k^r$.

1. If T is bounded with positive denominator, then so is T'.
2. If T is refining, then so is T'.
3. If T has a monotonicity type, e.g., $(\uparrow, \uparrow)$, then T' has the same type.

The proofs of these statements are analogous to the corresponding ones for matrices.

6.2 Emission of Result Digits

Of course, absorption is not enough; we also need a method to *emit* digits of the output stream representing the result of a computation.

Emission from Matrices. Let M be a matrix and ξ a digit stream. To emit a digit k of $\langle M\rangle([\xi]_r)$, we must transform this value into the form $[k : \eta]_r = \langle A^r_k\rangle([\eta]_r)$. This can be done by writing M as product $A^r_k \cdot M'$ for some matrix M'. The equation $M = A^r_k \cdot M'$ yields $M' = A^{r*}_k \cdot M$ using the inverse of A^r_k. Thus, emission is performed by $M(\xi) = k : (A^{r*}_k \cdot M)(\xi)$.

An explicit formula for the product $A^{r*}_k \cdot M$ is obtained by specialising (1):

$$A^{r*}_k \cdot M \;=\; \begin{pmatrix} r & -k \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a & c \\ b & d \end{pmatrix} \;=\; \begin{pmatrix} ra - kb & rc - kd \\ b & d \end{pmatrix} \tag{19}$$

Of course, we cannot emit an arbitrary digit. If the output stream is to begin with k, then the result of the computation should be in the corresponding digit interval $[k]_r$; otherwise the method would be unsound. Thus, we can only emit k from $M(\xi)$ if we know that its value is contained in $[k]_r$. Without looking into ξ, we know nothing about it. Thus, the condition $M(\xi) \in [k]_r$ must hold for all digit streams ξ, which is equivalent to $M(I_0) \subseteq [k]_r$.

– Emission: $M(\xi) = k : (A^{r*}_k \cdot M)(\xi)$.
  This operation is permitted only if $M(I_0) \subseteq [k]_r$.

For the following invariance properties, let $M' = A^{r*}_k \cdot M$.

1. If M is bounded with positive denominator, then so is M'.
   Proof: This is obvious since M and M' have the same denominator.
2. If M is refining and the emission leading to M' was permitted, then M' is refining again. Proof: If $M(I_0) \subseteq [k]_r = A^r_k(I_0)$, then $M'(I_0) = A^{r*}_k(M(I_0)) \subseteq A^{r*}_k(A^r_k(I_0)) = I_0$.
3. If M is increasing (decreasing), then so is M'.
   Proof: M' is M composed with the increasing function A^{r*}_k.

Emission from Tensors. Emission from a tensor works similar to emission from a matrix:

– Emission: $T(\xi, \eta) = k : (A^{r*}_k \cdot T)(\xi, \eta)$.
  This operation is permitted only if $T(I_0^2) \subseteq [k]_r$.

An explicit formula for the product $A^{r*}_k \cdot T$ is obtained by specialising (7):

$$\begin{pmatrix} r & -k \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \;=\; \begin{pmatrix} ra - kb & rc - kd & re - kf & rg - kh \\ b & d & f & h \end{pmatrix} \tag{20}$$

This variant of emission satisfies invariance properties 1–3 analogous to those for matrices.

6.3 Sketch of an Algorithm

We are now able to sketch an algorithm for applying a refining 1-LFT given by a matrix M to a digit stream:

Algorithm 1
Let $M_0 = M$. *Then for every* $n \geq 0$ *do:*
If there is a digit k *such that* $M_n(I_0) \subseteq [k]_r$,
then output digit k *and let* $M_{n+1} = A_k^{r*} \cdot M_n$,
else read the next digit k *from the input stream and let* $M_{n+1} = M_n \cdot A_k^r$.

The matrices M_0, M_1, etc. represent the internal state of the algorithm. Hence, we refer to them collectively as the *state matrix.* (In an imperative language, they would all occupy the same variable.)

For tensors an additional problem comes up: if no emission is possible, should we absorb a digit from the left argument or from the right? A simple strategy is to alternate between left and right absorption, while a more sophisticated strategy could look into the tensor to see which absorption is more likely to lead to a subsequent emission.

6.4 The Emission Conditions

Algorithm 1 was not very specific on how to find a digit k such that the image of the LFT is contained in $[k]_r$, or to find out that such a digit does not exist. These questions will be handled for base $r = 2$ only since this case allows for a simple solution: try the 3 possibilities $k = \mathbf{1}, \mathbf{0}, \bar{\mathbf{1}}$ in turn. (An idea of what to do for a general base can be obtained by looking at Section 9.3 below.)

The actual computation is simplified if we know some properties of the state matrix (or tensor) in question. Remember that some LFT properties are preserved by absorptions and permitted emissions. Thus, if the initial matrix is refining and bounded with positive denominator, then so will be all state matrices encountered in Alg. 1. Moreover, if the initial matrix has some specific monotonicity property, then all state matrices will have this property. Thus, for the following, we always assume a refining bounded matrix with positive denominator, and we shall try to exploit monotonicity as far as possible.

Base 2: Matrices. Let M be a refining bounded matrix with positive denominator. First, we consider the case that M is increasing, so that $M(I_0) = [M(-1), M(1)]$. Since M is refining, we know $M(I_0) \subseteq I_0$, or $M(-1) \geq -1$ and $M(1) \leq 1$. Then $M(I_0) \subseteq [\mathbf{1}]_2 = [0, 1]$ iff $M(-1) \geq 0$ and $M(1) \leq 1$, where the second condition is redundant. The first condition reads $\frac{-a+c}{-b+d} \geq 0$. Since the denominator is positive, this is equivalent to $a \leq c$. Similarly, $M(I_0) \subseteq [\bar{\mathbf{1}}]_2 = [-1, 0]$ iff $M(-1) \geq -1$ and $M(1) \leq 0$, where the first condition is redundant. The second condition reads $\frac{a+c}{b+d} \leq 0$. Since the denominator is positive, this is equivalent to $-a \geq c$.

Finally, $M(I_0) \subseteq [\mathbf{0}]_2 = [-\frac{1}{2}, \frac{1}{2}]$ iff $M(-1) \geq -\frac{1}{2}$ and $M(1) \leq \frac{1}{2}$, where no condition is redundant. The first condition reads $\frac{-a+c}{-b+d} \geq -\frac{1}{2}$, or $2(c-a) \geq b-d$. The second condition reads $\frac{a+c}{b+d} \leq \frac{1}{2}$, or $2(c+a) \leq b+d$. Checking these two conditions becomes more efficient if they contain common subexpressions that can be evaluated ahead. Indeed, the first condition can be transformed into

$b - 2c \le d - 2a$, and the second into $2c - b \le d - 2a$. Hence, the two conditions may be even combined into one, namely $|2c - b| \le d - 2a$.

If M is decreasing, the roles of $M(1)$ and $M(-1)$ are interchanged. This means that in the emission conditions, a and b have to be replaced by $-a$ and $-b$, respectively, while c and d remain unchanged. Thus, the condition $a \le c$ for emission of $\mathbf{1}$ becomes $-a \le c$, the condition $-a \ge c$ for emission of $\bar{\mathbf{1}}$ becomes $a \ge c$, and finally, the condition $|2c - b| \le d - 2a$ becomes $|2c + b| \le d + 2a$. All conditions are summarised in the following table:

Type	$\mathbf{1}$	$\bar{\mathbf{1}}$	$\mathbf{0}$
$\uparrow$	$a \le c$	$-a \ge c$	$\lvert 2c - b\rvert \le d - 2a$
$\downarrow$	$-a \le c$	$a \ge c$	$\lvert 2c + b\rvert \le d + 2a$

Since the condition for $\mathbf{0}$ is more complicated than the other two, we propose to check the conditions in the order $\mathbf{1}$, $\bar{\mathbf{1}}$, $\mathbf{0}$. This has the additional advantage that there is a situation where some tests can be avoided because they are bound to fail. Suppose the checks of the emission conditions for $\mathbf{1}$ and $\bar{\mathbf{1}}$ both failed, but the check for $\mathbf{0}$ succeeded. Then the digit $\mathbf{0}$ is emitted, and the current matrix $\left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ is replaced by $\left(\begin{smallmatrix} 2a & 2c \\ b & d \end{smallmatrix}\right)$ according to (19). Yet the relationship between $\pm 2a$ and $2c$ is the same as between $\pm a$ and c, which means that the emission conditions for $\mathbf{1}$ and $\bar{\mathbf{1}}$ will again fail; therefore, only the condition for $\mathbf{0}$ needs to be checked again.

Base 2: Tensors. Now let T be a refining bounded tensor with positive denominator. First, we consider the case that T is of type $(\uparrow, \uparrow)$, so that $T(I_0^2) = [T(-1,-1), T(1,1)]$. Then $T(I_0^2) \subseteq [\mathbf{1}]_2 = [0,1]$ iff $T(-1,-1) \ge 0$; the other condition $T(1,1) \le 1$ holds anyway since T is refining. The relevant condition reads $\frac{a-c-e+g}{b-d-f+h} \ge 0$. Since the denominator is positive, this is equivalent to $c+e \le g+a$. Similarly, $T(I_0^2) \subseteq [\bar{\mathbf{1}}]_2 = [-1,0]$ iff $T(1,1) = \frac{a+c+e+g}{b+d+f+h} \le 0$, which is equivalent to $c + e \le -(g + a)$.

Finally, $T(I_0^2) \subseteq [\mathbf{0}]_2 = [-\frac{1}{2}, \frac{1}{2}]$ iff $T(-1,-1) \ge -\frac{1}{2}$ and $T(1,1) \le \frac{1}{2}$. The first condition reads $\frac{a-c-e+g}{b-d-f+h} \ge -\frac{1}{2}$, or $-2(a - c - e + g) \le b - d - f + h$. The second condition reads $\frac{a+c+e+g}{b+d+f+h} \le \frac{1}{2}$, or $2(a + c + e + g) \le b + d + f + h$. The first condition can be transformed into $d + f - 2a - 2g \le h + b - 2c - 2e$, and the second into $2a + 2g - d - f \le h + b - 2c - 2e$. Again, these two conditions can be combined into one, namely $|2(g + a) - (d + f)| \le (h + b) - 2(c + e)$. Note that $g + a$ and $c + e$ also occur in the tests for $\mathbf{1}$ and $\bar{\mathbf{1}}$; they need only be evaluated once.

If T is of type $(\uparrow, \downarrow)$ instead, then $T(-1,-1)$ must be replaced by $T(-1,1)$, and $T(1,1)$ by $T(1,-1)$. This corresponds to negation of a, b, e, f, while the other four parameters are unchanged. The other two monotonicity types can be handled by similar negations. The results are collected in the following table:

Type	$\mathbf{1}$	$\bar{\mathbf{1}}$	$\mathbf{0}$
$(\uparrow,\uparrow)$	$c+e \le g+a$	$c+e \le -(g+a)$	$\lvert 2(g+a)-(d+f)\rvert \le (h+b)-2(c+e)$
$(\uparrow,\downarrow)$	$c-e \le g-a$	$c-e \le -(g-a)$	$\lvert 2(g-a)-(d-f)\rvert \le (h-b)-2(c-e)$
$(\downarrow,\uparrow)$	$e-c \le g-a$	$e-c \le -(g-a)$	$\lvert 2(g-a)-(f-d)\rvert \le (h-b)-2(e-c)$
$(\downarrow,\downarrow)$	$-c-e \le g+a$	$-c-e \le -(g+a)$	$\lvert 2(g+a)+(d+f)\rvert \le (h+b)+2(c+e)$

If T is of unknown monotonicity type or does not have any type at all, then the conjunction of the four conditions in each column must be considered. The four conditions for $\mathbf{1}$ can be combined into the two conditions $|c+e| \le g+a$ and $|c-e| \le g-a$, and similarly for $\bar{\mathbf{1}}$, while no simplification seems to be possible in case of $\mathbf{0}$.

Again, the conditions for $\mathbf{0}$ are more complicated then the other two. If the order $\mathbf{1}$, $\bar{\mathbf{1}}$, $\mathbf{0}$ is chosen, then as in the matrix case, $\mathbf{1}$ and $\bar{\mathbf{1}}$ need not be checked again after emission of $\mathbf{0}$.

6.5 Examples

Example 6.1. Let's first consider the matrix $M = \left(\begin{smallmatrix} 3 & 0 \\ 0 & 4 \end{smallmatrix}\right)$ which means multiplication by $\frac{3}{4}$. The d-entry 4 is positive, and the determinant 12 is positive as well. The function is bounded ($d = 4 > |b| = 0$), and refining ($[M(-1), M(1)] = [-\frac{3}{4}, \frac{3}{4}] \subseteq I_0$). Therefore, we can use the emission conditions in the $\uparrow$ row of the matrix table. Generally, we check the conditions in the order $\mathbf{1}$, $\bar{\mathbf{1}}$, $\mathbf{0}$, except after emission of $\mathbf{0}$, where the conditions for $\mathbf{1}$ and $\bar{\mathbf{1}}$ are skipped because they are known to fail as pointed out above. We also take any opportunity to cancel common factors of the four parameters of the state matrix. Let's assume the digit sequence denoting the argument starts with $\mathbf{101}$.

Start: $M = \left(\begin{smallmatrix} 3 & 0 \\ 0 & 4 \end{smallmatrix}\right)$
$a \le c \Leftrightarrow 3 \le 0$ fails, $-a \ge c \Leftrightarrow -3 \ge 0$ fails, $|2c-b| \le d-2a \Leftrightarrow 0 \le -2$ fails.
Absorb 1 and set M to $\left(\begin{smallmatrix} 3 & 3 \\ 0 & 8 \end{smallmatrix}\right)$.
$a \le c \Leftrightarrow 3 \le 3$ succeeds.
Emit 1 and set M to $\left(\begin{smallmatrix} 6 & -2 \\ 0 & 8 \end{smallmatrix}\right)$. **Cancel** a factor of 2 so that $M = \left(\begin{smallmatrix} 3 & -1 \\ 0 & 4 \end{smallmatrix}\right)$.
$a \le c \Leftrightarrow 3 \le -1$ fails, $-a \ge c \Leftrightarrow -3 \ge -1$ fails, $|2c-b| \le d-2a \Leftrightarrow 2 \le -2$ fails.
Absorb 0 and set M to $\left(\begin{smallmatrix} 3 & -2 \\ 0 & 8 \end{smallmatrix}\right)$.
$a \le c \Leftrightarrow 3 \le -2$ fails, $-a \ge c \Leftrightarrow -3 \ge -2$ fails, $|2c-b| \le d-2a \Leftrightarrow 4 \le 2$ fails.
Absorb 1 and set M to $\left(\begin{smallmatrix} 3 & -1 \\ 0 & 16 \end{smallmatrix}\right)$.
$a \le c \Leftrightarrow 3 \le -1$ fails, $-a \ge c \Leftrightarrow -3 \ge -1$ fails,
but $|2c-b| \le d-2a \Leftrightarrow 2 \le 10$ succeeds.
Emit 0 and set M to $\left(\begin{smallmatrix} 6 & -2 \\ 0 & 16 \end{smallmatrix}\right)$. **Cancel** a factor of 2 so that $M = \left(\begin{smallmatrix} 3 & -1 \\ 0 & 8 \end{smallmatrix}\right)$.
$|2c-b| \le d-2a \Leftrightarrow 2 \le 2$ succeeds.
Emit 0 and set M to $\left(\begin{smallmatrix} 6 & -2 \\ 0 & 8 \end{smallmatrix}\right)$. **Cancel** a factor of 2 so that $M = \left(\begin{smallmatrix} 3 & -1 \\ 0 & 4 \end{smallmatrix}\right)$.
$|2c-b| \le d-2a \Leftrightarrow 2 \le -2$ fails.

Now, we should absorb a new digit, but we only assumed the prefix $\mathbf{101}$ to be known. Thus, the algorithm transforms the argument prefix $\mathbf{101}$ into the result prefix **100**. Note that $[\mathbf{101}] = [\frac{1}{2}, \frac{3}{4}]$ and $\mathbf{100} = [\frac{3}{8}, \frac{5}{8}] \supseteq [\frac{3}{8}, \frac{9}{16}] = M([\frac{1}{2}, \frac{3}{4}])$, as it should be. In practice, a demand for more output digits will automatically generate a demand for more input digits, which will be computed by the process computing the argument.

Example 6.2. Let's consider another example which involves something more complicated than multiplication by $\frac{3}{4}$, namely computing $\frac{1}{x+2}$. In contrast to $\frac{3}{4}x$, it is not immediate how a digit stream for $\frac{1}{x+2}$ can be computed from a digit stream for x. Yet the algorithm developed above provides the answer.

The function $x \mapsto \frac{1}{x+2}$ is a 1-LFT with matrix $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 2 \end{smallmatrix}\right)$. The entry $d = 2$ is positive, but the determinant -1 is negative. The function is bounded ($d = 2 > |b| = 1$), and refining ($[M(1), M(-1)] = [\frac{1}{3}, 1] \subseteq I_0$). Thus, the algorithm can be applied—with the emission conditions from the $\downarrow$ row of the table for matrices.

Start: $M = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 2 \end{smallmatrix}\right)$
$-a \le c \Leftrightarrow 0 \le 1$ succeeds. **Emit 1** and set M to $\left(\begin{smallmatrix} -1 & 0 \\ 1 & 2 \end{smallmatrix}\right)$.
$-a \le c \Leftrightarrow 1 \le 0$ fails, $a \ge c \Leftrightarrow -1 \ge 0$ fails, $|2c + b| \le d + 2a \Leftrightarrow 1 \le 0$ fails.
Absorb 1 and set M to $\left(\begin{smallmatrix} -1 & -1 \\ 1 & 5 \end{smallmatrix}\right)$.
$-a \le c \Leftrightarrow 1 \le -1$ fails, $a \ge c \Leftrightarrow -1 \ge -1$ succeeds.
Emit $\bar{\mathbf{1}}$ and set M to $\left(\begin{smallmatrix} -1 & 3 \\ 1 & 5 \end{smallmatrix}\right)$.
$-a \le c \Leftrightarrow 1 \le 3$ succeeds. **Emit 1** and set M to $\left(\begin{smallmatrix} -3 & 1 \\ 1 & 5 \end{smallmatrix}\right)$.
$-a \le c \Leftrightarrow 3 \le 1$ fails, $a \ge c \Leftrightarrow -3 \ge 1$ fails, $|2c + b| \le d + 2a \Leftrightarrow 3 \le -1$ fails.
Absorb 0 and set M to $\left(\begin{smallmatrix} -3 & 2 \\ 1 & 10 \end{smallmatrix}\right)$.
$-a \le c \Leftrightarrow 3 \le 2$ fails, $a \ge c \Leftrightarrow -3 \ge 2$ fails, $|2c + b| \le d + 2a \Leftrightarrow 5 \le 4$ fails.
Absorb 1 and set M to $\left(\begin{smallmatrix} -3 & 1 \\ 1 & 21 \end{smallmatrix}\right)$.
$-a \le c \Leftrightarrow 3 \le 1$ fails, $a \ge c \Leftrightarrow -3 \ge 1$ fails, $|2c + b| \le d + 2a \Leftrightarrow 3 \le 15$ succeeds.
Emit 0 and set M to $\left(\begin{smallmatrix} -6 & 2 \\ 1 & 21 \end{smallmatrix}\right)$.
$|2c + b| \le d + 2a \Leftrightarrow 5 \le 9$ succeeds. **Emit 0** and set M to $\left(\begin{smallmatrix} -12 & 4 \\ 1 & 21 \end{smallmatrix}\right)$.
$|2c + b| \le d + 2a \Leftrightarrow 9 \le -3$ fails.

Thus, the algorithm maps the input prefix **101**, which denotes the interval $[\frac{1}{2}, \frac{3}{4}]$, into the output prefix $\mathbf{1\bar{1}100}$, which denotes the interval $[\frac{11}{32}, \frac{13}{32}]$. This interval really contains $M([\frac{1}{2}, \frac{3}{4}]) = [\frac{4}{11}, \frac{2}{5}]$ as it should be.

Note that in Example 6.1, a common factor of 2 could occasionally be cancelled, while in Example 6.2, no cancellation was possible. We will return to this point in Section 8.1. Note further the way in which absorptions (A) and emissions (E) alternate. In the first example, the sequence is AEAAEE, and in the second, it is EAEEAAEE. In both cases, the next would be an A. There appears to be some randomness in these sequences, but it is not too bad; there seem to be no strings of 3 consecutive A's or E's.

The question how many absorptions are needed to achieve a certain number of emissions is important for the performance of the algorithm. We would not like situations where a large number of absorptions is needed before the next emission is possible. The worst possibility were a situation where the algorithm keeps on absorbing for ever without ever being able to emit something (like in the problem of computing $3 \cdot 0.333\cdots$ in ordinary decimal notation). Fortunately, we can prove that this cannot happen; apart from some finite start-up phase in the beginning, absorptions and emissions will approximately alternate. This will be shown in the next section.

7 Contractivity and Expansivity

Our next goal is to derive bounds for the number of absorptions that are required to achieve a certain number of emissions. Such bounds can be obtained from bounds of the derivative(s) of the LFT. In fact, we are able to obtain theoretical bounds for an even larger class of functions.

7.1 Functions of One Argument

Let I be an interval (as always closed) and $F : I \to \mathbb{R}$ a C^1-function, i.e., a continuous function which is differentiable with continuous derivative F'. The *mean value theorem* of analysis states that for all x, y in I, there is some z between x and y (hence in I) such that $F(x) - F(y) = F'(z) \cdot (x - y)$. This property gives bounds for the length of the interval $F(I)$. First, we have for $I = [u, v]$

$$\ell(F(I)) \;\geq\; |F(v) - F(u)| \;\geq\; \inf_{z \in I} |F'(z)| \cdot (v - u) \;=\; \exp^I F \cdot \ell(I) \tag{21}$$

where $\exp^I F = \inf_{z \in I} |F'(z)|$ is the *expansivity* of F on I.

Second, we have

$$\ell(F(I)) \;=\; \sup_{x,y \in I} |F(x) - F(y)| \;\leq\; \sup_{z \in I} |F'(z)| \cdot \sup_{x,y \in I} |x - y| \;=\; \mathrm{con}^I F \cdot \ell(I) \tag{22}$$

where $\mathrm{con}^I F = \sup_{z \in I} |F'(z)|$ is the *contractivity* of F on I. Since $F' : I \to \mathbb{R}$ is continuous, the contractivity is always finite, and so we have $0 \;\leq\; \exp^I F \;\leq\; \mathrm{con}^I F \;<\; \infty$.

Together with Prop. 2.1, the bounds derived above will provide information about possible emissions. Assume F is a C^1-function defined on the base interval $I_0 = [-1, 1]$ with $F(I_0) \subseteq I_0$. We now look for theoretical lower and upper bounds for the number of digits required from a digit stream ξ representing an argument x if we want to compute a certain number n of digits of a stream representing the result $F(x)$. We work with a general base $r \geq 2$.

If a prefix δ of length m of the argument stream ξ is known, then x is in the interval $I = [\delta]_r$ of length $\ell(I) = 2r^{-m}$. Hence, $F(x)$ is in the interval $F(I)$, whose length l is bounded by $\exp^I F \cdot 2r^{-m} \leq l \leq \mathrm{con}^I F \cdot 2r^{-m}$. The dependence on the actual interval I can be removed by replacing $\exp^I F$ by $\exp^{I_0} F \leq \exp^I F$, and $\mathrm{con}^I F$ by $\mathrm{con}^{I_0} F \geq \mathrm{con}^I F$. Dropping the index I_0, we obtain $\exp F \cdot 2r^{-m} \leq l \leq \operatorname{con} F \cdot 2r^{-m}$. (Yet note for later that we may work with $\exp^J F$ and $\operatorname{con}^J F$ instead, if we are interested in arguments taken from a subinterval $J \subseteq I_0$.)

By Prop. 2.1, we know that (at least) n result digits can be emitted if $l \leq r^{-n}$. Hence, n digits can be emitted if $\operatorname{con} F \cdot 2r^{-m} \leq r^{-n}$, or $r^m \geq 2 \operatorname{con} F \cdot r^n$, or $m \geq \log_r(2 \operatorname{con} F) + n$. Thus, to emit n output digits, we need at most $\lceil \log_r(2 \operatorname{con} F) \rceil + n$ input digits. This statement even applies to the case $\operatorname{con} F = 0$, where the logarithm is $-\infty$. For, in this case, F is constant, and any number of output digits can be obtained without looking at the input at all.

By Prop. 2.1, we also know that $l \leq 2r^{-n}$ if (at least) n result digits can be emitted. Thus, $\exp F \cdot 2r^{-m} \leq 2r^{-n}$, or $m \geq \log_r(\exp F) + n$ if n result digits can be emitted. Hence, we need at least $\lceil \log_r(\exp F) \rceil + n$ input digits to obtain n result digits. In case of $\exp F = 0$ where the logarithm is $-\infty$, this statement still holds (trivially), but does not yield any useful information.

Theorem 7.1. *Let F be a C^1-function defined on the base interval I_0. To obtain n digits of $F(x)$ for x in I_0, one needs at least $c^< + n$ and at most $c^> + n$ digits of x, where*

$$c^< = \lceil \log_r(\exp F) \rceil \quad \textit{and} \quad c^> = \lceil \log_r(2 \operatorname{con} F) \rceil$$

where r is the base of the number system, $\exp F = \inf_{x \in I_0} |F'(x)|$ *and* $\operatorname{con} F = \sup_{x \in I_0} |F'(x)|$.

For functions with $\operatorname{con} F \geq \exp F > 0$, the theorem implies that asymptotically, the number of absorptions and emissions will be equal, i.e., on the long run and on average, one absorption is required for every emission. Locally, we see that for n emissions, at least $c^< + n$ absorptions are needed, while for $n+1$ emissions, at most $c^> + n + 1$ are required. Hence, after any emission, we need at most $c^> - c^< + 1$ absorptions, before the next emission is permitted. In particular, it can never happen that an infinite amount of absorptions does not lead to any emission.

For affine F, i.e., $F(x) = ax + b$, F' is constant and so $\exp F$ and $\operatorname{con} F$ coincide. In this case, the two bounds in Theorem 7.1 are close together: For base 2, they always differ by one, while for large bases, they are even identical in most cases, allowing the exact prediction of the number of required argument digits. For non-affine F, $\exp F$ and $\operatorname{con} F$ may differ considerably, leading to less accurate estimations.

Let's now consider the case that F is a 1-LFT which is bounded on I_0, given by a matrix $M = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ with non-negative d. Recall from Section 4 that M is C^1 with $M'(x) = \frac{\det M}{(bx+d)^2}$. From Prop. 5.2 and its proof, we know that $bx + d$ is positive for $x \in I_0$, with least value $d - |b|$. It is not hard to see that its largest value is $d + |b|$, and therefore

$$\operatorname{con} M \;=\; \frac{|\det M|}{(d - |b|)^2} \quad \text{and} \quad \exp M \;=\; \frac{|\det M|}{(d + |b|)^2}\,. \tag{23}$$

For affine matrices ($b = 0$), both expressions simplify to $\frac{|ad|}{d^2} = \frac{|a|}{d}$.

With these values, Theorem 7.1 not only describes the theoretical complexity of obtaining $M(x)$, but also the actual complexity of Algorithm 1. For, the algorithm detects an opportunity for emission as soon as it arises because its tests are logically equivalent to the emission condition.

In Example 6.1, we have $M = \left(\begin{smallmatrix} 3 & 0 \\ 0 & 4 \end{smallmatrix}\right)$, hence $\exp M = \operatorname{con} M = \frac{3}{4}$, and so $c^< = \lceil \log_2 \frac{3}{4} \rceil = 0$ and $c^> = \lceil \log_2 \frac{3}{2} \rceil = 1$. Hence, between n and $n{+}1$ absorptions are needed for n emissions, and the maximum number of absorptions between any two emissions is $1 - 0 + 1 = 2$.

In Example 6.2, we have $M = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 2 \end{smallmatrix}\right)$, hence $\exp M = \frac{1}{9}$ and $\operatorname{con} M = 1$, and so $c^< = \lceil \log_2 \frac{1}{9} \rceil = -3$ and $c^> = \lceil \log_2 2 \rceil = 1$. Hence, between $n - 3$ and $n + 1$ absorptions are needed for n emissions, and the maximum number of absorptions between any two emissions is $1 - (-3) + 1 = 5$.

Note that for 1-LFT's M, we have $\exp M = 0$ iff $\operatorname{con} M = 0$ iff $\det M = 0$ iff M is a constant function. Hence, there are only two cases: if $\det M \neq 0$, the number of absorptions and emissions is asymptotically equal, while for $\det M = 0$, any number of digits can be emitted without absorbing anything.

7.2 Functions of Two Arguments

Let I and J be two intervals (as always closed) and $F : I \times J \to \mathbb{R}$ a C^1-function, i.e., a continuous function which is differentiable in both arguments with continuous derivatives $\frac{\partial F}{\partial x}$ and $\frac{\partial F}{\partial y}$. Thus, for fixed x in I, $F|_x : J \to \mathbb{R}$ with $F|_x(y) = F(x, y)$ is a C^1-function on J, and for fixed y in J, $F|^y : I \to \mathbb{R}$ with $F|^y(x) = F(x, y)$ is a C^1-function on I.

Let's first derive a lower bound for $\ell(F(I, J))$. For every y in J, (21) implies

$$\ell(F(I \times J)) \;\geq\; \ell(F|^y(I)) \;\geq\; \exp^I(F|^y) \cdot \ell(I) \;\geq\; {\exp_{\mathrm{L}}}^{I,J} F \cdot \ell(I) \tag{24}$$

where

$${\exp_{\mathrm{L}}}^{I,J} F \;=\; \inf_{y \in J} \exp^I(F|^y) \;=\; \inf_{x \in I, y \in J} |\tfrac{\partial F}{\partial x}(x, y)| \tag{25}$$

is the left *expansivity* of F on $I \times J$. Dually, we have

$$\ell(F(I \times J)) \;\geq\; {\exp_{\mathrm{R}}}^{I,J} F \cdot \ell(J) \quad \text{where} \quad {\exp_{\mathrm{R}}}^{I,J} F \;=\; \inf_{x \in I, y \in J} |\tfrac{\partial F}{\partial y}(x, y)| \tag{26}$$

is the right *expansivity* of F on $I \times J$.

For an upper bound, consider $x_1, x_2 \in I$ and $y_1, y_2 \in J$. With (22), we obtain

$$\begin{aligned} |F(x_1, y_1) - F(x_2, y_2)| &\leq |F(x_1, y_1) - F(x_2, y_1)| + |F(x_2, y_1) - F(x_2, y_2)| \\ &\leq \operatorname{con}^I(F|^{y_1}) \cdot \ell(I) \;+\; \operatorname{con}^J(F|_{x_2}) \cdot \ell(J) \\ &\leq {\operatorname{con}_{\mathrm{L}}}^{I,J} F \cdot \ell(I) \;+\; {\operatorname{con}_{\mathrm{R}}}^{I,J} F \cdot \ell(J) \end{aligned} \tag{27}$$

where $\quad {\operatorname{con}_{\mathrm{L}}}^{I,J} F \;=\; \sup_{y \in J} \operatorname{con}^I(F|^y) \;=\; \sup_{x \in I, y \in J} |\tfrac{\partial F}{\partial x}(x, y)|$

and $\quad {\operatorname{con}_{\mathrm{R}}}^{I,J} F \;=\; \sup_{x \in I} \operatorname{con}^J(F|_x) \;=\; \sup_{x \in I, y \in J} |\tfrac{\partial F}{\partial y}(x, y)|.$

Note that these numbers are finite because the partial derivatives are continuous. Finally, Relation (27) yields $\ell(F(I \times J)) =$

$$\sup_{x_1,x_2 \in I} \sup_{y_1,y_2 \in J} |F(x_1,y_1) - F(x_2,y_2)| \le {\mathrm{con_L}}^{I,J}\, F \cdot \ell(I) + {\mathrm{con_R}}^{I,J}\, F \cdot \ell(J)\,. \quad (28)$$

Assume now F is a C^1-function defined on $I_0^2 = [-1,1] \times [-1,1]$ with $F(I_0^2) \subseteq I_0$. Assume further that $F(x_1,x_2)$ is to be computed where each x_i is given by a digit stream ξ_i, and we want to find out how many argument digits are needed to obtain n digits of the result $F(x_1,x_2)$.

If a prefix δ_i of length m_i of the argument stream ξ_i is known, then x_i is in the interval $I_i = [\delta_i]_r$ of length $\ell(I_i) = 2r^{-m_i}$. Hence, $F(x_1,x_2)$ is in the interval $F(I_1,I_2)$, whose length l is bounded by $l^< \le l \le l^>$, where

$$\begin{aligned} l^< &= \max\left({\mathrm{exp_L}}^{I_1,I_2}\, F \cdot 2r^{-m_1},\ {\mathrm{exp_R}}^{I_1,I_2}\, F \cdot 2r^{-m_2}\right) \\ l^> &= {\mathrm{con_L}}^{I_1,I_2}\, F \cdot 2r^{-m_1} + {\mathrm{con_R}}^{I_1,I_2}\, F \cdot 2r^{-m_2} \end{aligned}$$

Again, the dependence on the actual intervals I_1 and I_2 can be removed by enlarging both of them to I_0. We call the resulting bounds $l^\ll$ and $l^\gg$. For ease of notation, we drop the indices in ${\mathrm{exp_L}}^{I_0,I_0}$, etc.

By Prop. 2.1, we know that (at least) n result digits can be emitted if $l \le r^{-n}$, which is the case if $l^\gg \le r^{-n}$. Hence, n digits can be emitted if $\mathrm{con_L}\, F \cdot 2r^{-m_1} \le \frac{1}{2} r^{-n}$ and $\mathrm{con_R}\, F \cdot 2r^{-m_2} \le \frac{1}{2} r^{-n}$. The first condition is equivalent to $r^{m_1} \ge 4\,\mathrm{con_L}\, F \cdot r^n$, or $m_1 \ge \log_r(4\,\mathrm{con_L}\, F) + n$. Thus, to emit n output digits, $\lceil \log_r(4\,\mathrm{con_L}\, F)\rceil + n$ digits from the left argument and $\lceil \log_r(4\,\mathrm{con_R}\, F)\rceil + n$ digits from the right argument are sufficient.

By Prop. 2.1, we also know that $l \le 2r^{-n}$ if (at least) n result digits can be emitted. Thus, if n digits can be emitted, then $l^\ll \le 2r^{-n}$, or $\mathrm{exp_L}\, F \cdot 2r^{-m_1} \le 2r^{-n}$ and $\mathrm{exp_R}\, F \cdot 2r^{-m_2} \le 2r^{-n}$, or $m_1 \ge \log_r(\mathrm{exp_L}\, F) + n$ and $m_2 \ge \log_r(\mathrm{exp_R}\, F) + n$. These relations indicate how many digits from the two arguments are at least needed to obtain n result digits.

Theorem 7.2. *Let F be a C^1-function with two arguments defined on I_0^2. To obtain n digits in base r of $F(x_1,x_2)$ for x_1, x_2 in I_0, one needs at least $c_\mathrm{L}^< + n$ digits of x_1 and $c_\mathrm{R}^< + n$ digits of x_2, where*

$$c_\mathrm{L}^< = \lceil \log_r(\mathrm{exp_L}\, F)\rceil \quad \textit{and} \quad c_\mathrm{R}^< = \lceil \log_r(\mathrm{exp_R}\, F)\rceil\,.$$

On the other hand, $c_\mathrm{L}^> + n$ digits of x_1 and $c_\mathrm{R}^> + n$ digits of x_2 are sufficient to obtain (at least) n output digits, where

$$c_\mathrm{L}^> = \lceil \log_r(4\,\mathrm{con_L}\, F)\rceil \quad \textit{and} \quad c_\mathrm{R}^> = \lceil \log_r(4\,\mathrm{con_R}\, F)\rceil\,.$$

For functions with $\mathrm{exp_L}\, F > 0$ and $\mathrm{exp_R}\, F > 0$, the theorem implies that on the long run and on average, one absorption from each argument is required for every emission. Analogously to the case of one argument, one can show that it can never happen that an infinite amount of absorptions from both sides does not lead to any emission.

Unlike the case of matrices, there are no simple formulae for the left and right contractivities and expansivities of a general tensor. The reason is that the general forms of the partial derivatives are too complicated. Yet for some special tensors, concrete bounds can be obtained easily.

The tensor for addition is not refining, but $T = \left(\begin{smallmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{smallmatrix}\right)$ with $T(x, y) = \frac{1}{2}(x + y)$ is refining. Since $\frac{\partial T}{\partial x}(x, y) = \frac{\partial T}{\partial y}(x, y) = \frac{1}{2}$, we have $\exp_{\mathrm{L}} T = \exp_{\mathrm{R}} T = \mathrm{con}_{\mathrm{L}}\, T = \mathrm{con}_{\mathrm{R}}\, T = \frac{1}{2}$. Hence in base 2, at least $n - 1$ digits and at most $n + 1$ digits must be absorbed from both sides to obtain n output digits. (In practice, $n - 1$ digits are not sufficient.)

The tensor $T = \left(\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{smallmatrix}\right)$ with $T(x, y) = xy$ is refining. Since $\frac{\partial T}{\partial x}(x, y) = y$ and $\frac{\partial T}{\partial y}(x, y) = x$, we have $\exp_{\mathrm{L}} T = \exp_{\mathrm{R}} T = 0$ and $\mathrm{con}_{\mathrm{L}}\, T = \mathrm{con}_{\mathrm{R}}\, T = 1$. Hence in base 2, $n + 2$ digits from both sides are sufficient to obtain n output digits, but we do not get useful lower bounds. Indeed, we have $(\mathbf{0} : \xi) \cdot \eta = \mathbf{0} : (\xi \cdot \eta)$, and therefore, an arbitrary number of output digits can be obtained without looking at the second argument if the first argument is $\mathbf{0}^\omega$.

8 The Size of the Entries

When a non-singular refining matrix is applied to a digit stream, we know from Theorem 7.1 that between $c^< + 2n$ and $c^> + 2n$ *transactions* (absorptions plus emissions) are needed to obtain n output digits. At first glance, these transactions (and the emission tests) seem to require only constant time (see (16) and (19)), but we need to take into account the size of the four entries of the state matrix. In Example 6.2, the entries seem to grow during the course of the computation, and the time required by the integer operations in the transactions and tests (mainly addition and comparison) is linear in the bit size of the involved numbers. Thus, we should try to obtain bounds for the entries of the state matrix (or tensor) in order to obtain proper complexity results.

8.1 Common Factors

Cancellation of common factors of the entries of the state matrix could help to keep the entries small. In Example 6.1, a common factor of 2 could occasionally be cancelled, while there were no common factors at all in Example 6.2.

We first show that the range of possible common factors is quite limited.

Proposition 8.1. *Let M be a matrix or tensor in lowest terms (i.e., no non-trivial common factors in the entries), and let M' be the result of performing one transaction in base r (absorption or emission) at M. Then any common factor of M' divides r.*

Proof. Let $M = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ as usual. If M' results from absorbing digit k, then $M' = \left(\begin{smallmatrix} a & rc+ka \\ b & rd+kb \end{smallmatrix}\right)$. Any common factor g of a, b, $rc + ka$, and $rd + kb$ is also a common factor of ra, rb, rc, and rd. Since a, b, c, and d are relatively prime by assumption, g must divide r. The arguments for emission, where $M' = \left(\begin{smallmatrix} ra-kb & rc-kd \\ b & d \end{smallmatrix}\right)$, and for tensors are similar. $\square$

Even the limited amount of cancellation admitted by Prop. 8.1 does not show up in most cases. Note that without cancellation of common factors, neither absorption nor emission affect the b-entry of the state matrix or tensor. If b is odd like in Example 6.2, then it remains odd for ever, and there will never be any common factors in base 2. If b is even and non-zero, then common factors may occur, but only as often as the exponent of the largest power of 2 contained in b. After this amount of common factors has been cancelled out, the resulting value of b will be odd, and no further cancellations will be possible. Only if $b = 0$, an unlimited number of cancellations may occur. In the following two subsections, we study the two cases $b = 0$ and $b \neq 0$ for matrices more closely.

8.2 Affine Matrices

For an affine matrix ($b = 0$), the transactions simplify a bit:

$$\begin{pmatrix} a & c \\ 0 & d \end{pmatrix} \cdot A_k^r = \begin{pmatrix} a & rc + ka \\ 0 & rd \end{pmatrix} \qquad A_k^{r*} \cdot \begin{pmatrix} a & c \\ 0 & d \end{pmatrix} = \begin{pmatrix} ra & rc - kd \\ 0 & d \end{pmatrix} \tag{29}$$

Hence, the result of first absorbing k and then emitting l, or the other way round, is

$$A_l^{r*} \cdot \begin{pmatrix} a & c \\ 0 & d \end{pmatrix} \cdot A_k^r = \begin{pmatrix} ra & r^2c + rka - rld \\ 0 & rd \end{pmatrix} \tag{30}$$

which has a common factor of r. After cancelling it, we obtain $\left(\begin{smallmatrix} a & rc+ka-ld \\ 0 & d \end{smallmatrix}\right)$, which is the same as the original matrix, except for the c-entry. Similarly, we obtain a common factor r^k after performing k absorptions and k emissions in any order, and cancelling r^k will produce a matrix with the same a and d entries as the original one. The d-entry will only increase if there is an excess of absorptions over emissions; this increase consists of a factor of r for every additional absorption.

By Theorem 7.1, we know that at most $c^> + n$ absorptions are needed for n emissions. Thus, immediately before the last of these n emissions, $n-1$ emissions and at most $c^> + n$ absorptions have happened; the maximal possible excess is therefore $c^> + 1$. Recall $c^> = \lceil \log_r(2 \operatorname{con} M) \rceil = \lceil \log_r(2\frac{|a|}{d}) \rceil$. By Prop. 5.4, $|a| \leq d$ holds, whence $c^> \leq 1$. Therefore, the maximal possible excess of absorptions over emissions is 2.

Theorem 8.2. *Let $M_0 = \left(\begin{smallmatrix} a_0 & c_0 \\ 0 & d_0 \end{smallmatrix}\right)$ be an affine refining matrix with $d_0 \geq 0$, and $(M_n)_{n \geq 0}$ the sequence of matrices which results from Algorithm 1, with the additional provision that after each step, all common factors are cancelled out. Then all entries of M_n are bounded by $r^2 \cdot d_0$.*

This bound is sharp as can be seen from Example 6.1: The starting value is $d_0 = 4$, and so the theoretical upper bound is $2^2 \cdot 4 = 16$, which indeed occurs after four transactions. But Theorem 8.2 ensures that it cannot get worse.

Because of the constant upper bound in Theorem 8.2, the additions and comparisons needed to execute the algorithm take only constant time.

Corollary 8.3. *If an affine refining 1-LFT is applied to a digit stream, each transaction (absorption or emission) takes only constant time. Hence, n output digits can be computed in time $O(n)$.*

8.3 Non-Affine Matrices

Remember that b in $\left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ is invariant under absorptions and emissions. Hence in case $b \neq 0$, all common factors that may appear during the calculation are factors of b, and thus, cancellation of common factors can only lead to a constant size reduction. (In the special case $|b| = 1$, there will be no non-trivial common factors at all.)

Let us consider entry d, which is an upper bound for all other entries by Prop. 5.5. Emission does not affect d, while absorption of A_k^r transforms d into $d' = rd + kb$. Because of $|k| \leq r - 1$, one obtains $d' \leq rd + (r-1)|b|$ and $d' \geq rd - (r-1)|b|$, which lead to $d' + |b| \leq r(d + |b|)$ and $d' - |b| \geq r(d - |b|)$. These estimations can easily be iterated. Taking into account possible cancellations by common factors in the lower bound, one obtains:

Theorem 8.4. *Let $M_0 = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ be a refining matrix with $d \geq 0$ and $b \neq 0$, and let $M_m = \left(\begin{smallmatrix} a_m & c_m \\ b_m & d_m \end{smallmatrix}\right)$ be a matrix which results from M_0 by m absorptions in base r, any number of emissions, and cancellation of all common factors. Then $d_m \geq \frac{d-|b|}{|b|} r^m + 1$ and $d_m \leq (d + |b|)r^m - |b|$ holds (where the coefficients of r^m are positive by Prop. 5.2).*

For the matrix $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ of Example 6.2, we obtain in base 2 the estimations $2^m + 1 \leq d_m \leq 3 \cdot 2^m - 1$. For $m = 0, \ldots, 3$, the lower bounds are 2, 3, 5, 9, the upper bounds 2, 5, 11, 23, and the observed values of d_m are 2, 5, 10, 21, close to the upper bounds.

On the positive side, Theorem 8.4 ensures that the bit size of the d-entry (and with it all other entries by Prop. 5.5) is at most linear in the number of absorptions. On the negative side, it indicates that it really has linear bit size; the increase of the size of the d-entry cannot be avoided. The a- and c-entries may grow as well, but they need not, while b is guaranteed to remain small because it is invariant.

Theorem 8.4 also has a negative effect on efficiency. Remember (Theorem 7.1) that n emissions require $O(n)$ absorptions, and thus lead to a d-entry of bit size $O(n)$. The next execution of the loop in Algorithm 1 will thus need time $O(n)$ because it requires the calculation of either $2c - d$ (emission of $\mathbf{1}$), or $2c + d$ (emission of $\bar{\mathbf{1}}$), or $d - 2a$ (in the test whether $\mathbf{0}$ can be emitted). Therefore we obtain:

Theorem 8.5. *The calculation of the first n digits of the result of applying a non-affine refining 1-LFT to a digit stream needs time $O(n^2)$ if Algorithm 1 is used.*

8.4 Size Bounds for Tensors

For tensors, similar results hold, but their proofs are much more involved. Here, we present only the main results.

Proposition 8.6. *Let* $T_0 = \left(\begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix}\right)$ *be a refining tensor with* $h \geq 0$*, and let* T_m *be a tensor which results from* T_0 *by* m *absorptions in base* r*, any number of emissions, and cancellation of all common factors. Then all entries of* T_m *are bounded by* $r^m(h + |f| + |d| + |b|)$.

Proposition 8.7. *For every refining tensor* $T_0 = \left(\begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix}\right)$ *with* $h \geq 0$*, there is an integer* $m_0 \geq 0$ *such that after* $m \geq m_0$ *absorptions, any number of emissions,* but no cancellations*, the lower right entry* h' *of the resulting tensor satisfies* $h' \geq r^{m-m_0}$.

8.5 Cancellation in Tensors

From (17), (18), and (20), it follows that the entry b in $\left(\begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix}\right)$ is invariant under emissions and absorptions. Hence, only a finite amount of cancellation is possible if $b \neq 0$, and so, h will have size $\Theta(r^m)$ after m absorptions. Only in the case $b = 0$, an infinite amount of cancellations is possible. The result of emitting A_k^r from $\left(\begin{smallmatrix} a & c & e & g \\ 0 & d & f & h \end{smallmatrix}\right)$ is

$$\begin{pmatrix} ra & rc-kd & re-kf & rg-kh \\ 0 & d & f & h \end{pmatrix}.$$

The results of left and right absorption of A_k^r into $\left(\begin{smallmatrix} a & c & e & g \\ 0 & d & f & h \end{smallmatrix}\right)$ are the tensors

$$\begin{pmatrix} a & c & re+ka & rg+kc \\ 0 & d & rf & rh+kd \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} a & rc+ka & e & rg+ke \\ 0 & rd & f & rh+kf \end{pmatrix}.$$

These three tensors reveal that the three entries a, d, and f either remain the same or are multiplied by r. Hence—under the condition $b = 0$—the three conditions $a = 0$, $d = 0$, and $f = 0$ are invariant under absorptions and emissions, i.e., zeros at these positions will stay for ever. Yet the three tensors do not exhibit any opportunity for cancellation in themselves.

In the case of matrices, the opportunity for cancelling r appears only if an absorption and an emission are considered together. Analogously, we now consider the combined effect of absorbing k_1 from the left and k_2 from the right, and emitting l at $\left(\begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix}\right)$ (a *round*). The result, which does not depend on the temporal order of these three transactions, has a common factor of r in its 8 entries. Cancelling this factor leads to

$$\begin{pmatrix} a & rc+k_2a-ld & re+k_1a-lf & G \\ 0 & d & f & rh+k_1d+k_2f \end{pmatrix} \qquad (31)$$

where $G = r^2 g + rk_1c + rk_2e + k_1k_2a - rlh - k_1ld - k_2lf$.
Thus, in each round, a factor of r can be cancelled. Yet this is not enough: since a round contains two absorptions, the lower right entry increases by a factor of approximately r^2 in each round, i.e., with the cancellation, it still increases by approximately r. At least, it will be only half as big (in terms of bit size) as in the case $b \neq 0$. Note also that a, d, and f attain their original values after a round with cancellation. On the positive side, this means that these three entries are bounded, reducing both space and time complexity of the calculations. On the negative side, it implies that if at least one of these three values is non-zero, then only a finite amount of further cancellations is possible (none at all if at least one of a, d, f is 1 or -1). Thus, we may only hope for a further infinite amount of cancellations if $a = d = f = 0$. Under this assumption, there is indeed another common factor of r in Tensor (31). Its cancellation leads to

$$\begin{pmatrix} 0 & c & e & rg + k_1c + k_2e - lh \\ 0 & 0 & 0 & h \end{pmatrix} \tag{32}$$

Hence, the entries c, e, and h attain their original values. As h is the dominant entry, one may argue further as in the case of matrices that all entries are bounded during the calculation. Summarising, we have the following three cases for $\left(\begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix}\right)$ if all possible cancellations are performed:

1. If $b \neq 0$, then there are only finitely many cancellations possible. After m rounds, h has bit size $2m + O(1)$.
2. If $b = 0$, then it stays 0 for ever, and so do each of a, d, f in this case. If not all of a, d, f are zero, then a factor of r can be cancelled in each round, but apart from these, there are only finitely many cancellations possible. After m rounds, h has bit size $m + O(1)$.
3. If $b = a = d = f = 0$, then this remains true for ever, and a factor of r^2 can be cancelled in each round. All entries of the tensor have size $O(1)$.

Like in the case of matrices, these results imply that a calculation with a tensor T needs quadratic time, unless $b = a = d = f = 0$ or $\exp_{\mathrm{L}} T = 0$ or $\exp_{\mathrm{R}} T = 0$.

9 Handling Many Digits at Once

The complexity analysis given above has shown that apart from some exceptional cases, the computation of n output digits from $M(x)$ or $T(x, y)$ needs quadratic time $O(n^2)$—if Algorithm 1 is used which works digit by digit, handling each individual digit by a transaction. In this section, we show that handling many digits at once leads to a reduction in the complexity.

9.1 Multi-Digits

The key observation is that the product of two (and hence many) digit matrices is again a digit matrix, in a bigger base. The product of two digit matrices

$$\begin{pmatrix} 1 & k_1 \\ 0 & r_1 \end{pmatrix} \cdot \begin{pmatrix} 1 & k_2 \\ 0 & r_2 \end{pmatrix} = \begin{pmatrix} 1 & k_1r_2 + k_2 \\ 0 & r_1r_2 \end{pmatrix} \tag{33}$$

looks like a digit matrix again; indeed, the conditions $|k_i| \le r_i - 1$ imply

$$|k_1 r_2 + k_2| \;\le\; (r_1 - 1)\, r_2 + (r_2 - 1) \;=\; r_1 r_2 - 1$$

so that the result really is a digit matrix in base $r_1 r_2$. Iterating (33) yields

$$A^r_{k_1} \cdot \ldots \cdot A^r_{k_n} \;=\; A^R_K \quad \text{where} \quad R = r^n \;\text{ and }\; K = \sum_{i=1}^{n} k_i r^{n-i}\,. \tag{34}$$

Thus, instead of considering the digit sequence $k_1 \ldots k_n$, one may instead consider the single number K and the length n of the sequence. The number K with $|K| \le r^n - 1$ will be called an *n-multi-digit* in base r.

If a real number $x \in I_0$ is given, then we may ask for the first n digits of a possible digit stream representation of x; this request is written as $n?\,x$. According to the considerations above, we may accept that the answer is not given as a digit stream of length n, but as an n-multi-digit K. The number K with $|K| \le r^n - 1$ is a correct answer to the request $n?\,x$ iff x is in $[\frac{K-1}{r^n}, \frac{K+1}{r^n}]$. We write $K = n?\,x$ if K is a correct answer for $n?\,x$ (but note that there are usually two different correct answers, e.g., $1?\,\frac{1}{3}$ has the correct answers 0 and 1).

9.2 Multi-Digit Computation

Assume we are given a refining non-singular matrix M and an argument x in I_0, and we are asked for n digits of $M(x)$ in base 2. Theorem 7.1 provides two integers $c^<$ and $c^>$ such that at least $c^< + n$ and at most $c^> + n$ digits from x are needed to obtain n digits of $M(x)$. Thus we must ask for some number m of digits of x, but we only know $c^< + n \le m \le c^> + n$. There are two strategies that can be used:

1. Ask for $m = c^< + n$ digits from x and let $K = m?\,x$. Absorb $A^{2^m}_K$ into M and check whether n digits can be emitted from the resulting matrix M'. If yes, then do the emission, but if not, absorb one more digit from x, check again, etc. Alternatively, one may determine the number $c'^<$ belonging to M' and ask for $c'^< - c^<$ more digits from x, absorb these new digits into M' and check again whether n digits can be emitted, etc.
2. Ask for $m = c^> + n$ digits from x and let $K = m?\,x$. Absorb $A^{2^m}_K$ into M and emit n digits from the resulting matrix (which is guaranteed to be possible).

Strategy (1) ensures that as few as possible digits are read from x, but it is algorithmically more involved than strategy (2) since it involves checking whether the emission is possible, and if this fails, either degenerates to the old digit-by-digit algorithm, or involves finding out how many more argument digits are at least needed. Here, we shall follow strategy (2), which is easier to describe.

Assume $M = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ is given where the four entries are small. We need to determine $c^> \;=\; \lceil \log_2(2 \operatorname{con} M) \rceil \;=\; \lceil \log_2(\frac{2|\det M|}{(d-|b|)^2}) \rceil$. The rounded logarithm can be obtained by counting how often the denominator $(d - |b|)^2$ must be doubled until it is bigger than the numerator, or the other way round, depending on which is bigger in the beginning. Alternatively, the calculation may be based on bit sizes. Clearly, it is sufficient to compute $c^{>}$ once for M to serve several requests $n?\,M(x)$ with different n and x.

To handle a request $n?\,M(x)$, we compute $m = c^{>} + n$ and ask for $K = m?\,x$. Then we absorb $A_K^{2^m}$ into M:

$$M \cdot A_K^{2^m} \;=\; \begin{pmatrix} a & c \\ b & d \end{pmatrix} \cdot \begin{pmatrix} 1 & K \\ 0 & 2^m \end{pmatrix} \;=\; \begin{pmatrix} a & 2^m c + Ka \\ b & 2^m d + Kb \end{pmatrix} \tag{35}$$

Since the original entries are assumed to be small and 2^m and K have a bit size of $O(m) = O(n)$, the computations in (35) can be done in linear time $O(n)$. Let the result be $M' = \left(\begin{smallmatrix} a & C \\ b & D \end{smallmatrix}\right)$ with small a and b, and big C and D.

The next step is to find a suitable integer L with $|L| \leq 2^n - 1$ such that $A_L^{2^n}$ can be emitted from M', which is possible iff $M'(I_0) \subseteq [\frac{L-1}{2^n}, \frac{L+1}{2^n}]$. If M and hence M' are increasing, then $M'(I_0) = [\frac{C-a}{D-b}, \frac{C+a}{D+b}]$, and if M is decreasing, then $M'(I_0) = [\frac{C+a}{D+b}, \frac{C-a}{D-b}]$. Anyway, we know what $M'(I_0)$ is. In the next subsection, we shall show how to determine a suitable L from this information.

Before we come to this, we consider the case of tensors. Assume we are given a refining tensor T and two arguments x_1 and x_2 in I_0, and we are asked to compute the first n digits of a representation of the result $T(x_1, x_2)$. Although we did not show how to do this, it is in principle possible to compute the two integers $c_{\mathrm{L}}^{>} = \lceil \log_2(4\,\mathrm{con}_{\mathrm{L}}\, T) \rceil$ and $c_{\mathrm{R}}^{>} = \lceil \log_2(4\,\mathrm{con}_{\mathrm{R}}\, T) \rceil$ from Theorem 7.2. Then we may request $m_1 = c_{\mathrm{L}}^{>} + n$ digits from x_1 and $m_2 = c_{\mathrm{R}}^{>} + n$ digits from x_2 which will be delivered as multi-digits $K_1 = m_1?\,x_1$ and $K_2 = m_2?\,x_2$. Absorbing these multi-digits into $T = \left(\begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix}\right)$ yields

$$T' \;=\; \begin{pmatrix} a & C & E & G \\ b & D & F & H \end{pmatrix} \;=\; T \;\textcircled{\scriptsize L}\; A_{K_1}^{2^{m_1}} \;\textcircled{\scriptsize R}\; A_{K_2}^{2^{m_2}}$$

where T' is given by

$$\begin{pmatrix} a & 2^{m_2}c + K_2 a & 2^{m_1}e + K_1 a & 2^{m_1+m_2}g + 2^{m_1}K_2 e + 2^{m_2}K_1 c + K_1 K_2 a \\ b & 2^{m_2}d + K_2 b & 2^{m_1}f + K_1 b & 2^{m_1+m_2}h + 2^{m_1}K_2 f + 2^{m_2}K_1 d + K_1 K_2 b \end{pmatrix} \tag{36}$$

In contrast to the matrix case, we do not get away with a linear computation. The product $K_1 K_2$ is a product of two n-bit integers which needs time $\psi(n) > O(n)$. Currently, the best known algorithms yield $\psi_0(n) = O(n \log n \log\log n)$, but many software packages for big integer arithmetic come up with a multiplication which needs more time than $\psi_0(n)$, but is still more efficient than $O(n^2)$.

Apart from the product $K_1 K_2$, all other operations, including multiplication by the powers of 2, can be performed in linear time $O(n)$. Thus we still have linear time if $a = b = 0$; in this case, some power of 2 may be cancelled.

Again, the next step is to find a suitable L with $|L| \leq r^n - 1$ such that $A_L^{r^n}$ can be emitted from T', which is possible iff $T'(I_0^2) \subseteq [\frac{L-1}{2^n}, \frac{L+1}{2^n}]$. The two end points of $T'(I_0^2)$ are the smallest and the largest of the four corner values

$T'(\pm 1, \pm 1)$, respectively. If the monotonicity type of T and hence of T' is known, then it is clear which of the corner values are the smallest and the largest.

9.3 Multi-Digit Emission

The treatment in the previous section has left us with the following problem: given an integer $n > 0$ and a rational interval $[u, v] \subseteq I_0$, which arose as $M'(I_0)$ or $T'(I_0^2)$, find an integer L such that $|L| \leq 2^n - 1$ and $[u, v] \subseteq [\frac{L-1}{2^n}, \frac{L+1}{2^n}]$. Because we used the upper bounds for absorption, we know that such an L exists, but for the sake of generality, we also derive a condition for the existence of L.

The interval inclusion above can be written as $u \geq \frac{L-1}{2^n}$ and $v \leq \frac{L+1}{2^n}$, which is equivalent to $2^n v - 1 \leq L \leq 2^n u + 1$. Since L is required to be an integer, this in turn is equivalent to $v' \leq L \leq u'$, where $v' = \lceil 2^n v - 1 \rceil$ and $u' = \lfloor 2^n u + 1 \rfloor$. Note that v' and u' are integers.

Thus, the following seems to be the appropriate method: Compute the integers v' and u'. If $v' > u'$, then the emission of n digits is not possible. Otherwise, any integer L with $v' \leq L \leq u'$ can be emitted, for instance $L = v'$ or $L = u'$.

There is one remaining difficulty though: as an n-multi-digit, the chosen integer L should satisfy $|L| \leq 2^n - 1$. Yet if $u = 1$, then $u' = 2^n + 1$, and if $1 - 2^{-n} \leq u < 1$, then $u' = 2^n$; in both cases, the choice $L = u'$ is forbidden. Similarly, $v' = -2^n - 1$ or $v' = -2^n$ may happen if $v \leq -1 + 2^{-n}$, rendering the choice $L = v'$ unsuitable.

These problems may be solved as follows: remember $u \geq -1$, whence $u' \geq -2^n + 1$. Hence, u' is a suitable choice if $u' \leq 2^n - 1$. This condition can be expressed in terms of u as follows:

$$u' = \lfloor 2^n u + 1 \rfloor \leq 2^n - 1 \iff 2^n u + 1 < 2^n \iff u < 1 - 2^{-n} . \qquad (37)$$

Since $n > 0$, this is certainly the case if $u \leq 0$. Analogously, one may show that v' is suitable if $v \geq 0$. Since $u \leq v$, one of these two conditions is always satisfied. Actually, the decision which of u' and v' to take can be based on the sign of any element $w \in [u, v]$; for, $w \leq 0$ implies $u \leq 0$, and $w \geq 0$ implies $v \geq 0$.

Algorithm 2
Input: *An integer $n > 0$ and a rational interval $[u, v] \subseteq I_0$.*
Output: *An n-multi-digit L which can be emitted, or the information that such a digit does not exist.*
Method:
$u' = \lfloor 2^n u + 1 \rfloor$; $v' = \lceil 2^n v - 1 \rceil$;
if $u' < v'$ *then no such digit exists*
else if $w \geq 0$ *then* $L = v'$ *else* $L = u'$
(where w is any convenient test value from $[u, v]$).

This algorithm is sufficient to deal with the various cases of LFT's which have been handled in the previous section. Since we followed strategy (2) and absorbed sufficiently many digits to guarantee the emission, the test $u' < v'$ can be omitted.

In the matrix case, we have $[u, v] = M'(I_0)$ where $M' = \begin{pmatrix} a & C \\ b & D \end{pmatrix}$. If M' is increasing, then $u = \frac{C-a}{D-b}$ and $v = \frac{C+a}{D+b}$. A simple test value w in-between is $M'(0) = C/D$. The test $C/D \geq 0$ is equivalent to $C \geq 0$ since $D > 0$ by our general assumption. Hence, we obtain the following algorithm (which also includes the absorption phase):

Algorithm 3
Input: *A refining increasing matrix* $M = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ *with* $c^>$, *an argument* x, *and the desired number* $n > 0$ *of output digits.*
Output: *An* n*-multi-digit* $L = n?\,M(x)$.
Method:
$m = c^> + n;$
if $m > 0$ *then* $K = m?\,x;\quad C = 2^m c + Ka;\quad D = 2^m d + Kb$
else $C = c;\ D = d;$
if $C \geq 0$ *then* $L = \left\lceil \frac{2^n(C+a)}{D+b} \right\rceil - 1$ *else* $L = \left\lfloor \frac{2^n(C-a)}{D-b} \right\rfloor + 1$.

For a decreasing matrix, the algorithm has to be suitably modified.

The two numbers $\left\lfloor \frac{2^n(C-a)}{D-b} \right\rfloor$ and $\left\lceil \frac{2^n(C+a)}{D+b} \right\rceil$ are obtained by integer divisions of the $2n$-bit integers $2^n(C \pm a)$ by the n-bit integers $D \pm b$, resulting in n-bit integers. The complexity of such a division is the same as the complexity $\psi(n)$ of multiplying two n-bit integers. Apart from the two divisions, all other operations, including multiplication by 2^n, can be performed in linear time $O(n)$. Thus, we have managed to decrease the time needed to obtain n output digits from $O(n^2)$ (for non-affine matrices) to $\psi(n)$.

But what about affine matrices ($b = 0$) where the single digit algorithm already performed in time $O(n)$? Well, if $b = 0$, the fractions $\frac{2^n(C \pm a)}{D \pm b}$ simplify to $\frac{2^n(C \pm a)}{2^m d}$, where a power of 2 can be cancelled before the quotients are computed. After cancellation, these are divisions of an n bit integer by a small integer which can be done in linear time $O(n)$.

For tensors of known monotonicity type, similar variants of the general Algorithm 2 can be developed. For general tensors, the algorithm becomes more complicated.

10 Algebraic Operations

These are the basic arithmetic operations like addition and multiplication. For each operation, we shall show how exponents can be handled, and how its action on mantissas can be implemented by LFT's. The general algorithm for multi-digits (Alg. 2) can be specialised to the various cases (here only shown for addition). These specialised multi-digit operations will not depend on LFT's any more. Later, we consider transcendental functions like exponential and logarithm, where LFT's will be indispensable.

10.1 Addition $x_1 + x_2$

Exponents. If both arguments happen to have the same exponent, it can be taken out since $2^e x_1 + 2^e x_2 = 2^e(x_1 + x_2)$. If the exponents are different, then the smaller one can be increased because of $[\xi]_2 = 2^e[\mathbf{0}^e : \xi]_2$. If the exponents have been successfully handled, we are left with adding the mantissas. Unfortunately, the base interval $[-1, 1]$ is not closed under addition, but writing $x_1 + x_2$ as $2(x_1 \oplus x_2)$ with $x_1 \oplus x_2 = \frac{x_1+x_2}{2}$ solves the problem. Hence the exponent handling can be done as follows:

$$(e_1 \,\|\, \xi_1) + (e_2 \,\|\, \xi_2) \;=\; (e + 1 \,\|\, (\mathbf{0}^{e-e_1} : \xi_1) \oplus (\mathbf{0}^{e-e_2} : \xi_2)) \quad \text{where } e = \max(e_1, e_2).$$

Single-digit algorithm. The operation '$\oplus$' is a refining 2-LFT $T = \left(\begin{smallmatrix} \mathit{0} & 1 & 1 & 0 \\ \mathit{0} & \mathit{0} & \mathit{0} & 2 \end{smallmatrix}\right)$ of type $(\uparrow, \uparrow)$. By the analysis in Section 8.5 we know that the zeros written as $\mathit{0}$ are persistent, and that there are sufficient opportunities for cancellation so that the entries remain bounded. Thus, the single-digit algorithm for addition can be run with tensors of the form $\left(\begin{smallmatrix} \mathit{0} & c & e & g \\ \mathit{0} & \mathit{0} & \mathit{0} & h \end{smallmatrix}\right)$, i.e., four parameters which are small integers. Since the entries are bounded, only finitely many tensors may show up during the single-digit algorithm. Hence, the algorithm can be turned into the action of a finite state transducer operating on digits as pure symbols.

Multi-digit algorithm. Algorithm 2 can be adapted to the special case of addition. Because of $\frac{\partial T}{\partial x}(x, y) = \frac{\partial T}{\partial y}(x, y) = \frac{1}{2}$, we know $\mathrm{con}_{\mathrm{L}}\, T = \mathrm{con}_{\mathrm{R}}\, T = \frac{1}{2}$, whence $c_{\mathrm{L}}^{>} = c_{\mathrm{R}}^{>} = 1$. Thus, $n + 1$ digits from the two arguments are sufficient to obtain n result digits. With $K_i = (n + 1)?\,\xi_i$ for $i = 1, 2$, we have $u = \frac{K_1-1}{2^{n+1}} \oplus \frac{K_2-1}{2^{n+1}} = \frac{K_1+K_2-2}{2^{n+2}}$ and $v = \frac{K_1+K_2+2}{2^{n+2}}$. A convenient test value w in-between is $\frac{K_1+K_2}{2^{n+2}}$; the condition $w \geq 0$ is equivalent to $K_1 + K_2 \geq 0$. The two integer candidates are $u' = \lfloor 2^n u + 1 \rfloor = \lfloor (K_1 + K_2 + 2)/4 \rfloor$ and $v' = \lceil (K_1 + K_2 - 2)/4 \rceil$. Thus, the algorithm looks as follows:

> For $L = n?\,(\xi_1 \oplus \xi_2)$ do:
> $K_1 = (n + 1)?\,\xi_1$; $K_2 = (n + 1)?\,\xi_2$; $K = K_1 + K_2$;
> if $K \geq 0$ then $L = \lceil (K - 2)/4 \rceil$ else $L = \lfloor (K + 2)/4 \rfloor$.

Subtraction is very similar to addition and not included here.

10.2 Multiplication $x_1 * x_2$

Exponents: $\quad (e_1 \,\|\, \xi_1) * (e_2 \,\|\, \xi_2) \;=\; (e_1 + e_2 \,\|\, \xi_1 * \xi_2)$.

Zero digits. Multiplication '$*$' is a refining 2-LFT $T = \left(\begin{smallmatrix} 1 & 0 & 0 & 0 \\ \mathit{0} & \mathit{0} & \mathit{0} & 1 \end{smallmatrix}\right)$ which has no monotonicity type since $\xi_1 * \xi_2$ is increasing in ξ_1 for $\xi_2 \geq 0$, but decreasing for $\xi_2 \leq 0$. If ξ_2 starts with $\mathbf{1}$ or $\bar{\mathbf{1}}$, we are in one of these two cases, but $\mathbf{0}$ does not provide the necessary information. Yet we may push out any zero digits without bothering about monotonicity and without changing the state tensor: $(\mathbf{0} : \xi_1) * \xi_2 \;=\; \mathbf{0} : (\xi_1 * \xi_2)$, $\qquad \xi_1 * (\mathbf{0} : \xi_2) \;=\; \mathbf{0} : (\xi_1 * \xi_2)$.
This process requires only linear time in the number of emitted digits. It ends if enough digits have been emitted or both arguments are normalised.

Single-digit algorithm. If there are no more zero digits to be emitted, then the signs of the arguments can be read off from their first non-zero digits. From these signs, the monotonicity type to be used in the rest of the computation can be determined, e.g., $\xi_1 \geq 0$ and $\xi_2 \leq 0$ implies type $(\downarrow, \uparrow)$. By the analysis in Section 8.5 we know that $\left(\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{smallmatrix}\right)$ has three persistent zeros and belongs to the medium class of tensors that permit one cancellation in every round, which does not suffice to obtain bounded entries. The general form of the state tensor will be $\begin{pmatrix} a & C & E & G \\ 0 & 0 & 0 & H \end{pmatrix}$ with small a and big C, E, G, and H. The algorithm cannot be optimised to a finite state transducer, but some optimisations are possible because of the persistent zero entries. (Konecny [39] characterized the functions that can be computed by finite state transducers. Multiplication is not among these functions.)

10.3 Reciprocal $1/x$

This operation presents the difficulty that it is undefined for $x = 0$. To compute $1/x$, we first need to normalise the argument x by squeezing out zeros from the mantissa and reducing the exponent accordingly (see Section 2.3). This process does not terminate for $x = 0$ and may take very long for $x \approx 0$. A possible solution is to provide a lower bound for the exponent and to indicate a "potential division by 0" if this bound is reached.

If normalisation terminates, we know $x \neq 0$ and $\frac{1}{4} \leq |\xi| \leq 1$ for the final mantissa of x. Then $1 \leq \frac{1}{|\xi|} \leq 4$, whence $|\frac{1}{4\xi}| \leq 1$. This shows how to proceed after normalisation:

$$1/(e \parallel \xi) \;=\; (-e + 2 \parallel R(\xi)) \quad \text{where} \quad R(\xi) = \tfrac{1}{4\xi}$$

Function R is a 1-LFT, $R = \left(\begin{smallmatrix} 0 & 1 \\ 4 & 0 \end{smallmatrix}\right)$. It is decreasing, bounded and refining on the two intervals $[\frac{1}{4}, 1]$ and $[-1, -\frac{1}{4}]$. This is sufficient to use the single-digit algorithm for computing $R(\xi)$. Practically, this can be done by first absorbing the initial two digits of ξ, which are $\mathbf{11}$, $\mathbf{10}$, $\bar{\mathbf{1}}\mathbf{0}$, or $\bar{\mathbf{1}}\bar{\mathbf{1}}$ because of normalisation. Absorption of $\mathbf{11}$ leads to $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 3 \end{smallmatrix}\right)$ (after cancellation), absorption of $\mathbf{10}$ leads to $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ etc. These matrices are ordinary decreasing refining matrices as required by the single-digit algorithm ($\left(\begin{smallmatrix} 0 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ is exactly the matrix used in Example 6.2).

11 Infinite LFT Expressions

11.1 Infinite Matrix Products

We shall later see that many familiar constants like π or e can be written as (formal) infinite products $\prod_{n=0}^{\infty} M_n$ of matrices with integer entries. This is a generalisation of the infinite sequences (or products) of digit matrices that we have already seen. Moreover, functions like e^x can be realised as infinite products of matrices whose entries depend on the argument x.

Before we continue, we need to clarify what such an infinite product actually means. As finite products of matrices are again matrices, one should expect the same for an infinite product. The standard way to define the infinite product $\prod_{n=0}^{\infty} M_n$ would be that it is the limit of the finite products $\prod_{n=0}^{m} M_n$ as m goes to infinity. Yet such a definition would involve a notion of limit for matrices, or rather 1-LFT's. While it is not impossible to define such a limit notion, it is beyond the scope of these notes. To avoid this problem, we only define the results of applying infinite products to arguments (numbers or intervals); the product itself remains meaningless and is considered mainly as another way to present a sequence of matrices.

Given a real number argument y_0, it is straightforward to define $\prod_{n=0}^{\infty} M_n(y_0)$ as the limit of the sequence of real numbers $y_n = M_0 \cdots M_{n-1}(y_0)$, provided all the numbers y_n are well-defined (no division by 0) and the limit exists. Using this new notion, we obtain for instance the real number $y = \sum_{i=1}^{\infty} d_i 2^{-i}$ denoted by the digit stream $d_1 d_2 \cdots$ as $\prod_{n=1}^{\infty} A_{d_n}(0)$, because $A_{d_1} \cdots A_{d_n}(0) = (\sum_{i=1}^{n} d_i 2^{n-i})/2^n$ converges to y. Actually, the argument 0 can be replaced by any real number y_0 since $A_{d_1} \cdots A_{d_n}(y_0) = (y_0 + \sum_{i=1}^{n} d_i 2^{n-i})/2^n$ also converges to y.

Now we replace the argument y_0 by an interval J_0. In analogy to the number case we consider the intervals

$$J_n \;=\; M_0 \cdots M_{n-1}(J_0). \tag{38}$$

The sequence $(J_n)_{n \geq 0}$ of intervals is *nested* if $J_n \supseteq J_{n+1}$ for all $n \geq 1$ (this does not include the inclusion $J_0 \supseteq J_1$ which is disregarded deliberately). The inclusion $J_n \supseteq J_{n+1}$ means $M_0 \cdots M_{n-1}(J_0) \supseteq M_0 \cdots M_n(J_0)$. If the matrices $M_0, \ldots, M_{n-1}$ are non-singular, this is equivalent to $M_n(J_0) \subseteq J_0$. Therefore in the non-singular case, the sequence of intervals is nested iff all LFT's M_n with $n \geq 1$ are refining w.r.t. the interval J_0. Note that M_0 need not be refining, but it should be bounded on J_0 so that $J_1 = M_0(J_0)$ and all other intervals are well-defined.

Following these considerations, an infinite product $\prod_{n=0}^{\infty} M_n(J_0)$ is called *refining* if M_0 is bounded on J_0 and all M_n for $n \geq 1$ are refining for J_0. This includes the sequences of our signed number representation, where M_0 is an exponent matrix, which is bounded on the base interval I_0, and the remaining matrices are digit matrices, which are refining for I_0.

We say that the refining product $\prod_{n=0}^{\infty} M_n(J_0)$ has as value the real number y if the intersection of the nested sequence of intervals $J_1 \supseteq J_2 \supseteq \cdots$ is the singleton set $\{y\}$. For instance, the product $E_e \prod_{n=1}^{\infty} A_{d_n}(I_0)$ corresponding to the number representation $(e \parallel d_1 d_2 \ldots)$ has as value the real number $2^e \cdot \sum_{i=1}^{\infty} d_i 2^{-i}$ denoted by the representation.

Application of an infinite product to a number and to an interval are clearly related. If $y = \prod_{n=0}^{\infty} M_n(J_0)$, then also $y = \prod_{n=0}^{\infty} M_n(y_0)$ for all y_0 in J_0. On the other hand, the interval notion is more restricted and hence more powerful than the point notion because it includes the fact that the interval sequence is nested, which provides lower and upper bounds for all sequences $(y_n)_{n \geq 0}$ coming from arguments $y_0 \in J_0$.

11.2 Convergence Criteria

A nested sequence of intervals $J_n = [u_n, v_n]$ converges to some single point iff $\ell(J_n) = v_n - u_n \to 0$ as $n \to \infty$. This single point is then the common limit of $(u_n)_{n\geq 1}$ and $(v_n)_{n\geq 1}$. Because of this observation, a convergence criterion may be obtained from the notion of contractivity. Iterating Relation (22) yields

$$\ell(J_n) \;=\; \ell((M_0 \cdots M_{n-1})(J_0)) \;\leq\; \mathrm{con}^{J_0} M_0 \cdot \ldots \cdot \mathrm{con}^{J_0} M_{n-1} \cdot \ell(J_0)\,.$$

Thus, we obtain the following:

Theorem 11.1.
A refining infinite product $\prod_{n=0}^{\infty} M_n(J_0)$ converges if $\prod_{n=0}^{\infty} \mathrm{con}^{J_0} M_n = 0$.

Usually, we shall not directly apply this criterion, but the following corollary:

Corollary 11.2. *If $\prod_{n=0}^{\infty} M_n(J_0)$ is a refining infinite product with the property $\lim_{n\to\infty} \mathrm{con}^{J_0} M_n < 1$, then the product converges to a real number.*

11.3 Transformation of Infinite Products

(Formal) infinite products can be transformed by algebraic manipulation with the hope that the result of the transformation has better convergence properties than the original product.

Given $\prod_{n=0}^{\infty} M_n$, select a sequence $(U_n)_{n\geq 1}$ of *non-singular* matrices. Then finite products can be transformed as follows:

$$M_0 \cdots M_{n-1}\, U_n \;=\; M_0\, U_1\, U_1^*\, M_1\, U_2 \cdots U_{n-1}^*\, M_{n-1}\, U_n \;=\; \widetilde{M}_0\, \widetilde{M}_1 \cdots \widetilde{M}_{n-1} \tag{39}$$

using the new matrices

$$\widetilde{M}_0 \;=\; M_0\, U_1 \qquad \text{and} \qquad \widetilde{M}_n \;=\; U_n^*\, M_n\, U_{n+1} \quad \text{for } n \geq 1. \tag{40}$$

Because any infinite product $\prod_{n=0}^{\infty} M_n$ of non-singular matrices can be transformed into any other product $\prod_{n=0}^{\infty} \widetilde{M}_n$ by choosing $U_1 = M_0^* \widetilde{M}_0$ and $U_{n+1} = M_n^* U_n \widetilde{M}_n$, one needs separate arguments for the convergence of the new product to the same value as the old one.

If the original product is applied to a real number y_0, then its value is the limit of the sequence $y_n = M_0 \cdots M_{n-1}(y_0)$. If there is a real number $\widetilde{y}_0$ such that $U_n(\widetilde{y}_0) = y_0$ for all $n \geq 1$, then the number sequence induced by the new matrices at $\widetilde{y}_0$ is the same as the sequence induced by the old matrices at y_0 because of $\widetilde{M}_0 \cdots \widetilde{M}_{n-1}(\widetilde{y}_0) \;=\; M_0 \cdots M_{n-1} U_n(\widetilde{y}_0) \;=\; y_n$ using (39). Hence we obtain:

Proposition 11.3. *If $\prod_{n=0}^{\infty} \widetilde{M}_n$ results from transforming $\prod_{n=0}^{\infty} M_n$ with $(U_n)_{n\geq 1}$ and y_0 and $\widetilde{y}_0$ are two real numbers satisfying $U_n(\widetilde{y}_0) = y_0$ for all $n \geq 1$, then $\prod_{n=0}^{\infty} \widetilde{M}_n(\widetilde{y}_0) = \prod_{n=0}^{\infty} M_n(y_0)$ (this means, the first expression converges if and only if the second converges, and if they converge, they have the same value).*

11.4 Infinite Products from Taylor Series

We want to implement transcendental functions by infinite products, and so we need methods to obtain such products from more familiar representations. One such representation is the Taylor power series $f(x) = \sum_{n=0}^{\infty} a_n x^n$, e.g., $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$.

There are several ways to transform Taylor series into infinite products. Among the methods explored so far, the one described in the sequel turned out to be the most useful one for the intended applications [29]. It can be applied whenever $a_n \neq 0$ for $n \geq 1$ and uses the matrices

$$M_0 = \begin{pmatrix} a_1 x & a_0 + a_1 x \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad M_n = \begin{pmatrix} x & x \\ 0 & q_n \end{pmatrix} \quad \text{for } n \geq 1 \tag{41}$$

where $q_n = \frac{a_n}{a_{n+1}}$. To show that these matrices correspond to the Taylor series, we claim that their finite products have the following form (up to scaling):

$$P_n = M_0 \cdots M_{n-1} = \begin{pmatrix} a_n x^n & \sum_{i=0}^{n} a_i x^i \\ 0 & 1 \end{pmatrix} \tag{42}$$

This claim can be verified by induction. For $n = 1$, we have $P_1 = M_0$, which clearly has the claimed form. For the step from n to $n+1$, we compute $P_{n+1} =$

$$P_n M_n = \begin{pmatrix} a_n x^n & \sum_{i=0}^{n} a_i x^i \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x & x \\ 0 & q_n \end{pmatrix} = \begin{pmatrix} a_n x^{n+1} & a_n x^{n+1} + q_n (\sum_{i=0}^{n} a_i x^i) \\ 0 & q_n \end{pmatrix}$$

Dividing all four entries by $q_n \neq 0$ yields the required form because of $a_n / q_n = a_{n+1}$.

From (42), $P_n(0) = \sum_{i=0}^{n} a_i x^i$ follows. Hence, the product $\prod_{n=0}^{\infty} M_n(0)$ converges if and only if the Taylor series converges, and yields the desired value $\sum_{i=0}^{\infty} a_i x^i$.

Of course, we do not want to apply the product to the real number 0, but to the base interval $I_0 = [-1, 1]$ of our number representation. Clearly, all matrices M_n are bounded (under the assumption $a_n \neq 0$ for $n \geq 1$). The matrices $\left(\begin{smallmatrix} x & x \\ 0 & q_n \end{smallmatrix}\right)$ are refining iff $|x| + |x| \leq |q_n|$ (Prop. 5.4). Hence, $\prod_{n=0}^{\infty} M_n(I_0)$ is refining for $|x| \leq q/2$, where $q = \inf_{n \geq 1} |q_n|$. The contractivity of M_n is $|x|/|q_n| \leq |x|/q$, which is at most $1/2$ for $|x| \leq q/2$. Thus, for $|x| \leq q/2$, $\prod_{n=0}^{\infty} M_n(I_0)$ is a refining convergent product. Since I_0 contains 0, its value coincides with $\prod_{n=0}^{\infty} M_n(0) = \sum_{n=0}^{\infty} a_n x^n$ as desired.

11.5 Infinite Products from Continued Fractions

Another, less familiar source of infinite products are continued fraction expansions. A continued fraction is an infinite expression

$$a_0 + \cfrac{b_1}{a_1 + \frac{b_2}{\cdots}} \tag{43}$$

parameterised by numbers $(a_n)_{n\geq 0}$ and $(b_n)_{n\geq 1}$. It denotes the limit of the sequence of partial continued fractions

$$a_0, \quad a_0 + \frac{b_1}{a_1}, \quad a_0 + \frac{b_1}{a_1 + \frac{b_2}{a_2}}, \quad \ldots$$

provided that this limit exists. For ease of notation, the infinite expression (43) is written as $\langle a_0; b_1, a_1; b_2, a_2; \ldots\rangle$.

Like for Taylor series, there are several ways to turn a continued fraction into an infinite product. We use the following:

$$M_0 = \begin{pmatrix} 1 & a_0 \\ 0 & 1 \end{pmatrix} \qquad \text{and} \qquad M_n = \begin{pmatrix} 0 & b_n \\ 1 & a_n \end{pmatrix} \quad \text{for } n \geq 1. \tag{44}$$

Since $M_0(y) = a_0 + y$ and $M_n(y) = \frac{b_n}{a_n+y}$, the partial products $M_0 \cdots M_{n-1}(0)$ are exactly the partial continued fractions so that $\prod_{n=0}^{\infty} M_n(0)$ converges if and only if the continued fraction converges, and yields the same value.

In practical applications, this infinite product must usually be transformed into a more appropriate one before the argument can successfully be extended to the base interval I_0. Often, the transformation matrices are chosen as $U_n = \left(\begin{smallmatrix} 1 & 0 \\ 0 & u_n \end{smallmatrix}\right)$. Since 0 is a fixed point of these matrices ($U_n(0) = 0$), the transformed infinite product still has the value of the continued fraction when applied to 0 by Prop. 11.3. For the actual transformation, it is useful to note that

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & u_{n+1} \end{pmatrix} = \begin{pmatrix} a & c\,u_{n+1} \\ b & d\,u_{n+1} \end{pmatrix} \tag{45}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & u_n \end{pmatrix}^* \begin{pmatrix} a & c \\ b & d \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & u_{n+1} \end{pmatrix} = \begin{pmatrix} u_n\,a & u_n\,c\,u_{n+1} \\ b & d\,u_{n+1} \end{pmatrix} \tag{46}$$

11.6 The Evaluation of Infinite Products

Before we come to the implementation of the various transcendental functions by infinite products, we give hints on how to use the products in a practical implementation. For simplicity, we only consider refining products applied to the base interval $[-1, 1]$.

If all the matrices in $\prod_{n=0}^{\infty} M_n(I_0)$ have integer entries, there is a choice of several different evaluation algorithms. We only consider single digit approaches, but corresponding multi-digit realisations do exist. Generally, one has to assume that a matrix M_n can be created from its index n. First, the matrices may be put into a list which initially contains only M_0. In this list, each matrix absorbs the digits that are emitted from its right neighbour. Whenever the rightmost matrix M_n needs to absorb a digit, the next matrix M_{n+1} is created and appended to the list.

Second, the algorithm may be run with a state matrix which initially is M_0. Whenever the state matrix cannot emit a digit, it absorbs the next matrix M_n

down the list of matrices which has not been absorbed before. This next matrix is created on the fly from its index n. Thus, only one matrix must be stored (and the index of the next one to be absorbed), while in the first method, a whole list of matrices must be maintained. On the other hand, the upper bounds for space and time complexity of the ordinary single-digit algorithm do not hold here, since the matrices that are absorbed are usually much more complicated than the simple digit matrices.

Usually, the matrices in the infinite product depend on an argument x, like in the product derived from the Taylor series expansion. If the argument is a given rational, the matrices can be converted into integer matrices by suitable scaling, and we are back to the previous case. In the general case of an arbitrary real argument, this cannot be done; instead, the matrices must be converted into tensors. This is always possible if their four entries depend linearly on x, by using (6):

$$\begin{pmatrix} ax+e & cx+g \\ bx+f & dx+h \end{pmatrix} = \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix}\Bigg|_x$$

If T_n is the tensor belonging to M_n, the product $f(x) = \prod_{n=0}^{\infty} M_n(I_0)$ becomes the infinite tensor expression $f(x) = T_0(x, T_1(x, \ldots))$. Such an expression can only be evaluated by the first method indicated above: a list of tensors must be maintained which initially consists of T_0 only. Each tensor absorbs argument digits from the left, and from the right the digits emitted from the next tensor. If the last tensor needs a digit from its right argument, a new tensor is created and added to the list. This algorithm works if the tensors are sufficiently contractive so that (almost) each tensor needs strictly less than n digits from its right argument to emit n digits.

12 Transcendental Functions

12.1 Exponential Function

Argument reduction. The infinite products derived below will only behave well for $|x| \leq 1$, which is equivalent to the exponent of x being at most 0. Yet an arbitrary real argument can be brought into this region by exploiting the fact $\mathrm{e}^{2x} = (\mathrm{e}^x)^2$. Hence, an exponent $n \geq 0$ may be handled by $\mathrm{e}^{(n\|\xi)} = S^n(\mathrm{e}^{\xi})$ where S^n means n applications of the squaring operation S. (Admittedly, this can become quite inefficient for larger exponents.) Negative exponents $n < 0$ can be handled by putting the corresponding number of zero digits in front of the mantissa: $\mathrm{e}^{(n\|\xi)} = \mathrm{e}^{\xi'}$ where $\xi' = \mathbf{0}^{|n|} : \xi$.

Taylor series realisation. The well-known Taylor series for e^x is $\sum_{n=0}^{\infty} \frac{x^n}{n!}$. All coefficients $a_n = 1/n!$ are non-zero, so that the method of Section 11.4 can be applied. The quotient $q_n = a_n/a_{n+1}$ is $n+1$, so that $q = \inf_{n\geq 1} |q_n| = 2$. Thus we have

$$\mathrm{e}^x = \begin{pmatrix} x & x+1 \\ 0 & 1 \end{pmatrix} \prod_{n=1}^{\infty} \begin{pmatrix} x & x \\ 0 & n+1 \end{pmatrix} (I_0) \quad \text{for } |x| \leq 1. \tag{47}$$

All M_n with $n \geq 1$ have contractivity $\frac{|x|}{n+1} \leq \frac{1}{n+1}$.

As already mentioned, a representation such as (47) is open to two different interpretations. For rational arguments x, it is (equivalent to) an infinite product of integer matrices, e.g., $\mathrm{e} = \left(\begin{smallmatrix} 1 & 2 \\ 0 & 1 \end{smallmatrix}\right) \prod_{n=1}^{\infty} \left(\begin{smallmatrix} 1 & 1 \\ 0 & n+1 \end{smallmatrix}\right) (I_0)$. For general (real) arguments however, representation (47) should be turned into the infinite tensor expression $\mathrm{e}^x = T_0(x, T_1(x, T_2(x, \ldots)))$ with

$$T_0 \;=\; \begin{pmatrix} 1 & 1 & 0 & 1 \\ \mathit{0} & \mathit{0} & \mathit{0} & 1 \end{pmatrix} \quad \text{and} \quad T_n \;=\; \begin{pmatrix} 1 & 1 & 0 & 0 \\ \mathit{0} & \mathit{0} & \mathit{0} & n+1 \end{pmatrix}$$

where each tensor has 3 persistent zeros, indicated by $\mathit{0}$.

The tensors T_n for $n \geq 0$ realise the functions $T_n(x,y) = x(y+1)/(n+1)$ which are increasing in x because $y + 1 \geq 0$ for $y \in I_0$, but are not monotonic in y. They can be handled similar to multiplication: leading zero digits of x can be pushed out without changing the tensor, and then the first non-zero digit decides the monotonicity behaviour.

The front tensor $T_0(x,y) = x(y+1)+1$ has the same monotonicity behaviour, but cannot be handled immediately in the same way; notice also that it is bounded, but not refining, so that it must emit an exponent matrix first.

- For $x \in [0,1]$ (leading digit $\mathbf{1}$), T_0 has type $(\uparrow, \uparrow)$ and image $[1,3]$, so that the appropriate exponent is 2 (and $\mathbf{10}$ can be emitted after emitting the exponent matrix).
- For $x \in [-1,0]$ (leading digit $\bar{\mathbf{1}}$), T_0 has type $(\uparrow, \downarrow)$ and image $[-1,1]$, so that the appropriate exponent is 0.
- For $x \in [-\frac{1}{2}, \frac{1}{2}]$ (leading digit $\mathbf{0}$), T_0 has image $[0,2]$, so that the appropriate exponent is 1, and $\mathbf{1}$ can be emitted after the exponent matrix. The tensor resulting from these emissions is (up to scaling) $T_0' = \left(\begin{smallmatrix} 1 & 1 & 0 & 0 \\ \mathit{0} & \mathit{0} & \mathit{0} & 1 \end{smallmatrix}\right)$ with $T_0'(x,y) = x(y+1)$. Hence, all leading zeros of x can be pushed out without modifying T_0', and the first non-zero digit decides the monotonicity behaviour.

Continued fraction realisation. A continued fraction for the exponential function is

$$\mathrm{e}^x \;=\; \langle 1;\; x,\; 1 - \frac{x}{2};\; \frac{x^2}{16 \cdot 1^2 - 4},\; 1;\; \frac{x^2}{16 \cdot 2^2 - 4},\; 1;\; \ldots \rangle.$$

It corresponds to the product representation

$$\mathrm{e}^x \;=\; \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & x \\ 1 & 1 - x/2 \end{pmatrix} \prod_{n=1}^{\infty} \begin{pmatrix} 0 & x^2/(16n^2-4) \\ 1 & 1 \end{pmatrix} (0).$$

The product of the first two matrices is (up to scaling) $M_0 = \left(\begin{smallmatrix} 2 & 2+x \\ 2 & 2-x \end{smallmatrix}\right)$. The infinite product cannot directly be extended to the base interval $I_0 = [-1,1]$ since the denominators of the matrices $M_n = \left(\begin{smallmatrix} 0 & x^2/(16n^2-4) \\ 1 & 1 \end{smallmatrix}\right)$ become 0 at -1. This problem is solved by transforming the product with the matrices $U_n =$

$\begin{pmatrix} 1 & 0 \\ 0 & 4(2n-1) \end{pmatrix}$ $(n \geq 1)$, which have the form considered in Section 11.5. By (45), the new front matrix is

$$\widetilde{M}_0 \;=\; \begin{pmatrix} 2 & 4(2+x) \\ 2 & 4(2-x) \end{pmatrix} \;\cong\; \begin{pmatrix} 1 & 4+2x \\ 1 & 4-2x \end{pmatrix}$$

which is bounded on I_0 for $|x| \leq 1$. By (46), the other matrices are

$$\widetilde{M}_n \;=\; \begin{pmatrix} 0 & x^2 \frac{16(2n-1)(2n+1)}{4(2n-1)(2n+1)} \\ 1 & 4(2n+1) \end{pmatrix} \;=\; \begin{pmatrix} 0 & 4x^2 \\ 1 & 4(2n+1) \end{pmatrix}$$

These matrices are refining on I_0 for $|x| \leq 1$. By (23), the contractivity of $\widetilde{M}_n$ is $\frac{4x^2}{(8n+3)^2} \leq \frac{x^2}{16n^2}$ which is better (i.e., smaller) than the value $\frac{|x|}{n+1} \leq \frac{1}{n+1}$ achieved by the Taylor expansion. Therefore, $\prod_{n=0}^{\infty} \widetilde{M}_n(I_0)$ converges, and since $0 \in I_0$, it converges to $\prod_{n=0}^{\infty} \widetilde{M}_n(0) = \prod_{n=0}^{\infty} M_n(0) = \mathrm{e}^x$.

Like the Taylor product, the continued fraction product consists of integer matrices for rational arguments, e.g., $\mathrm{e} = \begin{pmatrix} 1 & 6 \\ 1 & 2 \end{pmatrix} \prod_{n=1}^{\infty} \begin{pmatrix} 0 & 4 \\ 1 & 8n+4 \end{pmatrix} (I_0)$. In contrast to the Taylor case, these matrices are not affine and hence more difficult to handle, but they have better contractivity.

For general (real) arguments, the representation must be turned into the infinite tensor expression $\mathrm{e}^x \;=\; T_0\,(x, T_1\,(x^2, T_2\,(x^2, \ldots)))$ which uses both x and x^2. The tensors are

$$T_0 \;=\; \begin{pmatrix} \mathit{0} & 2 & 1 & 4 \\ \mathit{0} & -2 & 1 & 4 \end{pmatrix} \quad \text{and} \quad T_n \;=\; \begin{pmatrix} \mathit{0} & 4 & 0 & 0 \\ \mathit{0} & \mathit{0} & 1 & 8n+4 \end{pmatrix}$$

where each tensor except T_0 has 3 persistent zeros, indicated by $\mathit{0}$. Taking into account that their left argument x^2 is ≥ 0, the tensors T_n for $n \geq 1$ have type $(\uparrow, \downarrow)$. Leading zero digits of x are doubled by squaring and can be pushed out of T_n without modifying it. Moreover, each T_n can emit $\mathbf{1}$ after reading $\mathbf{1}$ from x^2. The front tensor T_0 is a bit more complicated, but can be handled essentially like the front tensor of the Taylor expansion.

12.2 Logarithm

Definition. Natural logarithm $\ln x$ is the inverse of the exponential function. Thus it is only defined for arguments $x > 0$. To deal with negative arguments, we propose to actually compute the function $f(x) = \ln|x|$, which is the anti-derivative (indefinite integral) of the reciprocal function $1/x$.

Argument normalisation. Like the reciprocal itself, f is still undefined for 0. The handling of this special case is similar to the handling of $1/0$ in Section 10.3: To compute $f(x)$, we first normalise the argument x by squeezing out zeros from the mantissa and reducing the exponent accordingly (see Section 2.3). If normalisation terminates, we know $x \neq 0$ and $\frac{1}{4} \leq |\xi| \leq 1$ for the final mantissa of x. This mantissa will start with the digit $\mathbf{1}$ or $\bar{\mathbf{1}}$. The following description tells what to do in the positive case; the negative case is dual.

Argument reduction. Here, we use the fact that a (positive) normalised mantissa starts with **10** or **11**: $\ln(e \parallel \mathbf{1} : \xi) = \ln(2^e \cdot (\xi+1)/2) = (e-1)\cdot \ln 2 + \ln(1+\xi)$. For the constant $\ln 2$ see below. The first digit of ξ is **0** or **1**; hence $\xi \in [-\frac{1}{2}, 1]$.

Continued fraction expansion. A continued fraction for the function $\ln(1+x)$ is

$$\langle 0;\ x,\ 1;\ x/2,\ 1;\ v_1,\ 1;\ w_1,\ 1;\ v_2,\ 1;\ w_2,\ 1;\ \ldots \rangle$$

where $v_n = \frac{nx}{4n+2}$ and $w_n = \frac{(n+1)x}{4n+2}$. We now write this continued fraction as an infinite product like in Section 11.5, and immediately transform this product by $U_n = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 4 \end{smallmatrix}\right)$, using (45) and (46) in the step marked by '$\stackrel{\mathrm{T}}{=}$'. In the last step, the matrices are converted into tensors.

$$\begin{aligned}
\ln(1+x) &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & x \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & x/2 \\ 1 & 1 \end{pmatrix} \prod_{n=1}^{\infty} \begin{pmatrix} 0 & v_n \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & w_n \\ 1 & 1 \end{pmatrix} (0) \\
&= \begin{pmatrix} x & x \\ 1 & x/2+1 \end{pmatrix} \prod_{n=1}^{\infty} \begin{pmatrix} v_n & v_n \\ 1 & w_n + 1 \end{pmatrix} (0) \\
&= \begin{pmatrix} 2x & 2x \\ 2 & x+2 \end{pmatrix} \prod_{n=1}^{\infty} \begin{pmatrix} nx & nx \\ 4n+2 & (n+1)x + 4n + 2 \end{pmatrix} (0) \\
&\stackrel{\mathrm{T}}{=} \begin{pmatrix} 2x & 2x \cdot 4 \\ 2 & (x+2)\cdot 4 \end{pmatrix} \prod_{n=1}^{\infty} \begin{pmatrix} 4 \cdot nx & 4 \cdot nx \cdot 4 \\ 4n+2 & ((n+1)x + 4n + 2)\cdot 4 \end{pmatrix} (0) \\
&= \begin{pmatrix} x & 4x \\ 1 & 2x+4 \end{pmatrix} \prod_{n=1}^{\infty} \begin{pmatrix} 2nx & 8nx \\ 2n+1 & (2n+2)x + 8n + 4 \end{pmatrix} (0) \\
&= \begin{pmatrix} 1 & 4 & 0 & 0 \\ 0 & 2 & 1 & 4 \end{pmatrix}\Bigg|_x \prod_{n=1}^{\infty} \begin{pmatrix} 2n & 8n & 0 & 0 \\ 0 & 2n+2 & 2n+1 & 8n+4 \end{pmatrix}\Bigg|_x (0)
\end{aligned}$$

The tensors in the last line will be called T_n ($n \geq 0$), and the corresponding matrices in the second but last line M_n ($n \geq 0$). All tensors T_n exhibit one persistent zero, in the lower left corner. They are bounded on I_0^2 since the right entry in their second line is bigger than the sum of the two middle entries. The determinants of the matrices are $\det M_0 = 2x^2 \geq 0$ and $\det M_n = 4n(n+1)x^2 \geq 0$ for $n \geq 1$. Hence, all tensors are increasing in their second argument, for any x. For $y \in I_0$, one may also verify $\det(T_n|^y) \geq 0$, i.e., all the tensors have type $(\uparrow, \uparrow)$ in I_0^2. They are *not* refining for I_0, but remember that we only consider $x \in [-\frac{1}{2}, 1]$. For such x, the matrices M_n with $n \geq 1$ *are* refining. By (23), the contractivity of these matrices is

$$\frac{4n(n+1)x^2}{((2n+2)x + 6n + 3)^2} \quad \xrightarrow{n\to\infty} \quad \frac{4x^2}{(2x+6)^2} \;=\; \left(\frac{x}{x+3}\right)^2$$

For $x = -\frac{1}{2}, 0, \frac{1}{2}, 1$, this gives $\frac{1}{25}, 0, \frac{1}{49}, \frac{1}{16}$, respectively.

The constant $\ln 2$ can be derived from the general case as

$$\ln(1+1) \;=\; \begin{pmatrix} 1 & 4 \\ 1 & 6 \end{pmatrix} \prod_{n=1}^{\infty} \begin{pmatrix} 2n & 8n \\ 2n+1 & 10n+6 \end{pmatrix} (I_0)$$

with contractivity $\frac{1}{16}$ (in the limit). Another possibility is to exploit the fact $\ln 2 = -\ln(\frac{1}{2}) = -\ln(1-\frac{1}{2})$, which leads to a product with contractivity $\frac{1}{25}$ (in the limit):

$$\ln 2 \;=\; \begin{pmatrix} 1 & 4 \\ 2 & 6 \end{pmatrix} \prod_{n=1}^{\infty} \begin{pmatrix} -n & -4n \\ 2n+1 & 7n+3 \end{pmatrix} (I_0)\,.$$

Part II:
A Domain Framework for Computational Geometry

13 Introduction

In Part I we presented a framework for exact real number computation, where we developed a data type for real numbers and presented algorithms for computing elementary functions. We now turn our attention to computational geometry, where we are interested in computing geometric objects, such as lines, curves, planes, surfaces, convex hulls and Voronoi diagrams. In a broad sense, we can say that this represents an extension of exact arithmetic in that we now need to compute a subset of the Euclidean space rather than just a real number. In fact, the undecidability of comparison of real numbers in exact arithmetic has a close counterpart in computational geometry, namely the undecidability of the membership predicate for proper subsets of the Euclidean space. Thus, in computational geometry one has to deal with somewhat similar problems as in exact arithmetic. However, there are some other fundamental new issues which are not encountered in exact arithmetic, making computational geometry an independent subject of its own.

Computational geometry and solid modelling, as in Computer Aided Design (CAD), are fundamental in the design and manufacturing of all physical objects. However, these disciplines suffer from the lack of a proper and sound datatype. The current frameworks in these subjects are based, on the one hand, on discontinuous predicates and Boolean operations, and, on the other hand, on comparison of real numbers, which is undecidable. These essential foundations of the existing theory and implementations are both unjustified and unrealistic; they give rise to unreliable programs in practice.

Topology and geometry, as mainstream mathematical disciplines, have been developed to study continuous transformations on spaces. It is therefore an irony that the main building blocks in these subjects, namely the membership predicate of a set, the subset inclusion predicate, and the basic operations such as intersection are generally not continuous and therefore non-computable.

For example, in any Euclidean space $\mathbb{R}^n$ the membership predicate $\in_S$ of any subset $S \subseteq \mathbb{R}^n$ defined as

$$\begin{aligned} \in_S\colon \mathbb{R}^n &\to \{\mathsf{tt}, \mathsf{ff}\} \\ x &\mapsto \begin{cases} \mathsf{tt} \text{ if } x \in S \\ \mathsf{ff} \text{ if } x \notin S \end{cases} \end{aligned}$$

with the discrete topology on $\{\mathsf{tt}, \mathsf{ff}\}$ is continuous if and only if S is both open and closed, i.e. if S is either empty or the whole space. In fact, the membership predicate of any proper subset of $\mathbb{R}^n$ is discontinuous at the boundary of the subset.

Similarly, consider the intersection operator as a binary operator on the collection $\mathcal{C}(\mathbb{R}^n)$ of compact subsets of $\mathbb{R}^n$ equipped with the Hausdorff distance d_H defined on closed subsets by

$$d_H(C,D) \;=\; \max\,(\sup_{d\in D}\,\inf_{c\in C}|c-d|,\ \sup_{c\in C}\,\inf_{d\in D}|c-d|),$$

with the convention that $d_H(\emptyset,\emptyset)=0$ and for $C \neq \emptyset$, $d_H(\emptyset, C)=\infty$:

$$\begin{aligned} -\cap- : \mathcal{C}(\mathbb{R}^n) \times \mathcal{C}(\mathbb{R}^n) &\to \mathcal{C}(\mathbb{R}^n) \\ (A,B) &\mapsto A \cap B \end{aligned}$$

Then, $-\cap-$ is discontinuous whenever A and B just touch each other.

The non-continuity of the basic predicates and operations creates a foundational problem in computation, which has so far been essentially neglected. In fact, in order to construct a sound computational model for solids and geometry, one needs a framework in which these elementary building blocks are continuous and computable.

In practice, correctness of algorithms in computational geometry is usually proved using the Real RAM machine model of computation, in which comparison of real numbers is considered to be decidable. Since this model is not realistic, correct algorithms, when implemented, turn into unreliable programs.

A simple example is provided by computing, in any floating point format, first the intersection point x in the plane of two straight lines L_1 and L_2 meeting under a small angle, and then computing the minimum distance $d(x, L_1)$ and $d(x, L_2)$ from x to each of the two lines. In general, $d(x, L_1)$ and $d(x, L_2)$ are both positive and distinct.

A more sophisticated example is given by the implementation in floating point of any algorithm to compute the convex hull of a finite number of points in the plane. If there are three nearly collinear points A, B, C as in the picture, then depending upon the floating point format, the program can give, instead of the two edges AB and BC, any of the following:

(i) AB only.
(ii) AC only.
(iii) BC only.
(iv) none of them.

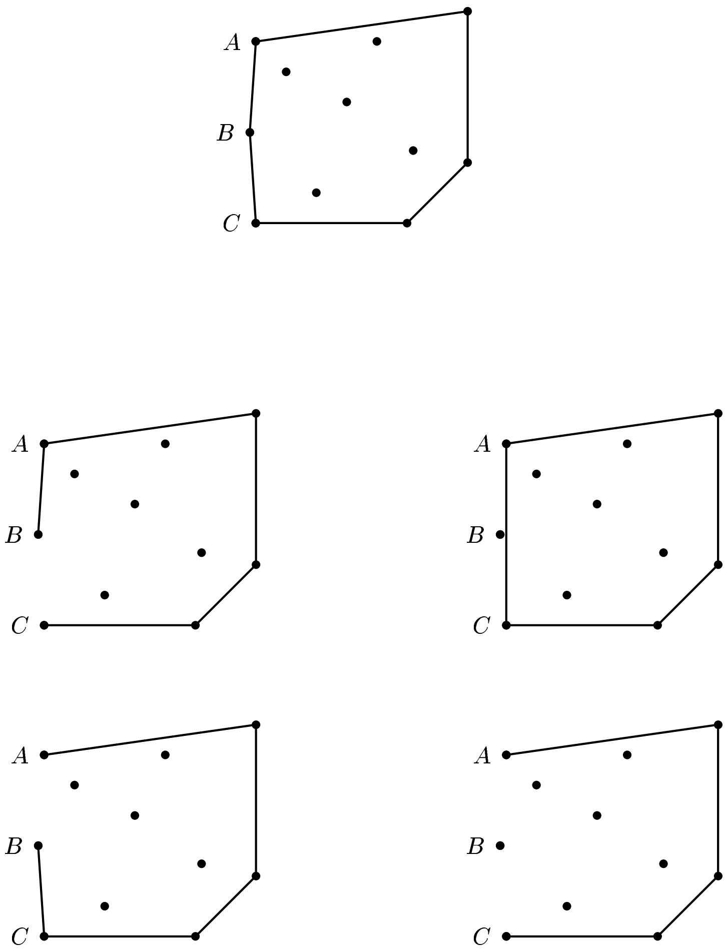

Fig. 1. The convex hull of a finite number of points (top picture) and four possible errors arising from floating point implementations.

In any of the above four cases, we get a logical inconsistency as the edges returned by the program do not give the correct convex hull and in the cases (i), (iii) and (iv) do not give a closed polygon at all.

In CAGD modelling operators, the effect of rounding errors on consistency and robustness of actual implementations is an open question, which is handled in industrial software by various heuristics.

The solid modelling framework provided by classical analysis, which allows discontinuous behaviour and comparison of exact real numbers, is not realistic as a model of our interaction with the physical world in terms of measurement and manufacturing. Nor is it realistic as a basis for the design of algorithms implemented on realistic machines, which can only deal with finite data. Industrial solid modelling software used for CAGD (Computer Aided Geometric Design), CAM (Computer Aided Manufacturing) or robotics is therefore infected by the disparity between the classical analysis paradigm and feasible computations. This disparity, as well as the representation of uncertainties in the geometry of the solid objects, is handled case by case, by various expensive and unsatisfactory "up to epsilon" ad-hoc heuristics. It is difficult, if at all possible, to improve and generalise these techniques, since their relatively poor success depends on the skill and experience of software engineers rather than on a well formalised methodology. In practice, the maintenance cost of some central geometric operators such as the Boolean operations or some specific variants of the Minkowski sum has always remained critical.

A robust algorithm is one whose correctness is proved with the assumption of a realistic machine model. Recursive analysis defines precisely what it means, in the context of the realistic Turing machine model of computation, to compute objects belonging to non-countable sets such as the set of real numbers.

Here, we use a domain-theoretic approach to recursive analysis to develop the foundation of an effective framework for solid modelling and computational geometry. It is based on the work of the second author with André Lieutier. In fact these notes form an abridged version of two papers [14, 15]; full details of proofs and many other results can be obtained from these papers.

We present the continuous domain of solid objects which gives a concrete model of computation on solids close to the actual practice of CAD engineers. In this model, the basic predicates, such as membership and subset inclusion, and operations, such as union and intersection, are continuous and computable. The set-theoretic aspects of solid modelling are revisited, leading to a theoretically motivated model. Within this model, some unavoidable limitations of solid modelling computations are shown and a sound framework to design specifications for feasible modelling operators is provided. Moreover, the model is able to capture the uncertainties of input data in actual CAD situations.

We need the following requirements for the mathematical model:

1. the notion of computability of solids has to be well defined,
2. the model has to reflect the observable properties of real solids,
3. it has to be closed under the Boolean operations and all basic predicates and operations have to be computable,

4. non-regular sets[1] have to be captured by the model as well as regular solids,
5. the model has to support a design methodology for actual robust algorithms.

A general methodology for the specification of feasible operators and the design of robust algorithms should rely on a sound mathematical model. This is why the domain-theoretic approach is a powerful framework both to model partial or uncertain data and to guide the design of robust software.

14 The Solid Domain

In this section, we introduce the solid domain, a mathematical model for representing rigid solids. The reader should refer to the Appendix for a basic introduction to the domain-theoretic notions required in the rest of this article. We focus here on the set-theoretic aspects of solid modelling. Our model is motivated by requirements 1 to 5 given above.

We first recall some basic notions in topology. For any subset A of a topological space X, the closure, $\overline{A}$, of A is the intersection of all closed sets containing A, the interior, A°, of A is the union of all open sets contained in A and the boundary, ∂A, of A is the set of points $x \in X$ such that any neighbourhood of x (i.e. any open set containing x) intersects both A and its complement A^c. Recall that an open set is *regular* if it is the interior of its closure; dually, a closed set is regular if it is the closure of its interior. The complement of a regular open set then is a regular closed set and vice versa. A subset $C \subseteq X$ is *compact* if for every collection of open subsets $\langle O_i \rangle_{i \in I}$ with $C \subseteq \bigcup_{i \in I} O_i$ there exists a finite set $J \subseteq I$ with $C \subseteq \bigcup_{i \in J} O_i$. A subset of $\mathbb{R}^d$ is compact iff it is bounded and closed.

Given any proper subset $S \subseteq \mathbb{R}^n$, the classical membership predicate $\in_S$: $\mathbb{R}^n \to \{\mathsf{tt}, \mathsf{ff}\}$ is continuous except on ∂S. Recall that a predicate is semi-decidable if there is an algorithm to confirm in finite time that it is true whenever the predicate is actually true. For example, membership of a point in an open set in $\mathbb{R}^n$ is semi-decidable, since if the point is given in terms of a shrinking sequence of rational rectangles, then in finite time one such rational rectangle will be completely inside the open set. On the other hand, if S is an open or closed set, then its boundary has empty interior and it is not semi-decidable that a point is on the boundary. For example if $n = 1$ and S is the set of positive numbers, then a real number $x \in \mathbb{R}$ is on the boundary of S iff $x = 0$ which is not decidable in computable analysis. It therefore makes sense from a computational viewpoint to redefine the membership predicate as the continuous function:

$$\begin{aligned} \in'_S \colon \mathbb{R}^n &\to \{\mathsf{tt}, \mathsf{ff}\}_\perp \\ x &\mapsto \begin{cases} \mathsf{tt} & \text{if } x \in S^\circ \\ \mathsf{ff} & \text{if } x \in S^{c\circ} \\ \perp & \text{otherwise.} \end{cases} \end{aligned}$$

[1] An open set is regular if it is the interior of its closure.

Here, $\{\mathsf{tt}, \mathsf{ff}\}_\perp$ is the three element poset with least element $\perp$ and two incomparable elements tt and ff. In the Scott topology $\{\mathsf{tt}\}$ and $\{\mathsf{ff}\}$ are open sets but $\{\perp\}$ is not open. We call this the *continuous membership predicate*. Then, two subsets, or two solid objects, are equivalent if and only if they have the same continuous membership predicate, i.e. if they have the same interior and the same exterior (interior of complement). By analogy with general set theory for which a set is completely defined by its membership predicate, we can define a solid object in $\mathbb{R}^n$ to be any continuous map of type $\mathbb{R}^n \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$. The definition of the solid domain is then consistent with requirement 1 since a computable membership predicate has to be continuous.

Note that a solid object, given by a continuous map $f : \mathbb{R}^n \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$, is determined precisely by two disjoint open sets, namely $f^{-1}(\mathsf{tt})$ and $f^{-1}(\mathsf{ff})$. Moreover, the interior $(f^{-1}(\mathsf{tt}) \cup f^{-1}(\mathsf{ff}))^{c\circ}$ of the complement of the union of these two open sets can be non-empty. If we now consider a second continuous function $g : \mathbb{R}^n \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$ with $f \sqsubseteq g$, then we have $f^{-1}(\mathsf{tt}) \subseteq g^{-1}(\mathsf{tt})$ and $f^{-1}(\mathsf{ff}) \subseteq g^{-1}(\mathsf{ff})$. This means that a more defined solid object has a larger interior and a larger exterior. We can think of the pair $f^{-1}(\mathsf{tt}), f^{-1}(\mathsf{ff})$ as the points of the interior and the exterior of a solid object as determined at some finite stage of computation. At a later stage, we obtain a more refined approximation g which gives more information about the solid object, i.e. more points of its interior and more points of its exterior.

Definition 14.1. *The* solid domain $(\mathbf{S}\mathbb{R}^n, \sqsubseteq)$ *of* $\mathbb{R}^n$ *is the set of ordered pairs* (A, B) *of disjoint open subsets of* $\mathbb{R}^n$ *endowed with the information order:*
$(A_1, B_1) \sqsubseteq (A_2, B_2) \iff A_1 \subseteq A_2$ *and* $B_1 \subseteq B_2$.

An element (A, B) of $\mathbf{S}\mathbb{R}^n$ is called a *partial solid*. The sets A and B are intended to capture, respectively, the interior and the exterior (interior of the complement) of a solid object, possibly, at some finite stage of computation. Note that $(\mathbf{S}\mathbb{R}^n, \sqsubseteq)$ is a directed complete partial order with $\bigsqcup_{i\in I}(A_i, B_i) = (\bigcup_{i\in I} A_i, \bigcup_{i\in I} B_i)$ and is isomorphic with the function space $\mathbb{R}^n \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$. By duality of open and closed sets, $(\mathbf{S}\mathbb{R}^n, \sqsubseteq)$ is also isomorphic with the collection of ordered pairs (A, B) of closed subsets of $\mathbb{R}^n$ with $A \cup B = \mathbb{R}^n$ with the information ordering: $(A_1, B_1) \sqsubseteq (A_2, B_2) \iff A_2 \subseteq A_1$ and $B_2 \subseteq B_1$.

Proposition 14.2. *The partial solid* $(A, B) \in (\mathbf{S}\mathbb{R}^n, \sqsubseteq)$ *is a maximal element iff* $A = B^{c\circ}$ *and* $B = A^{c\circ}$.

Proof. Let (A, B) be maximal. Since A and B are disjoint open sets, it follows that $A \subseteq B^{c\circ}$. Hence, $(A, B) \sqsubseteq (B^{c\circ}, B)$ and thus $A = B^{c\circ}$. Similarly, $B = A^{c\circ}$. This proves the "only if" part. For the "if" part, suppose that $A = B^{c\circ}$ and $B = A^{c\circ}$. Then, any proper open superset of A will have non-empty intersection with B and any proper open superset of B will have non-empty intersection with A. It follows that (A, B) is maximal. □

Corollary 14.3. *If* (A, B) *is a maximal element, then* A *and* B *are regular open sets. Conversely, for any regular open set* A*, the partial solid* $(A, A^{c\circ})$ *is maximal.*

Proof. For the first part, note that A is the interior of the closed set B^c and is, therefore, regular; similarly B is regular. For the second part, observe that $A^{c\circ c\circ} = (\overline{A})^\circ = A$. □

We define $(A, B) \in \mathbf{S}\mathbb{R}^n$ to be a *classical* solid object if $\overline{A} \cup \overline{B} = \mathbb{R}^n$.

Proposition 14.4. *Any maximal element is a classical solid object.*

Proof. Suppose (A, B) is maximal. Then $\mathbb{R}^n = A \cup \partial A \cup A^{c\circ} = \overline{A} \cup \overline{B}$, since $\overline{A} = A \cup \partial A$ and $A^{c\circ} \subseteq \overline{A^{c\circ}} = \overline{B}$. □

Classical solid objects form a larger family than the maximal elements, i.e. regular solids. For example, if $A = \{z \in \mathbb{R}^2 \mid |z| \leq 1\} \cup \{(x, 0) \in \mathbb{R}^2 \mid |x| \leq 2\}$, then A is represented in our model by the classical (non-regular) object (A°, A^c).

Theorem 14.5. *The solid domain* $(\mathbf{S}\mathbb{R}^n, \sqsubseteq)$ *is a bounded complete* ω*-continuous domain and* $(A_1, B_1) \ll (A_2, B_2)$ *iff* $\overline{A_1}$ *and* $\overline{B_1}$ *are compact subsets of* A_2 *and* B_2 *respectively.*

Proof. To characterise the way-below relation, first assume that $\overline{A_1}$ and $\overline{B_1}$ are compact subsets of A_2 and B_2 respectively. If $A_2 \subseteq \bigcup_{i\in I} U_i$ and $B_2 \subseteq \bigcup_{i \in I} V_i$, where the unions are assumed to be directed, then we get $\overline{A_1} \subseteq A_2 \subseteq \bigcup_{i\in I} U_i$ and $\overline{B_1} \subseteq B_2 \subseteq \bigcup_{i\in I} V_i$. By compactness of $\overline{A_1}$ and $\overline{B_1}$ it follows that there exists $i \in I$ with $\overline{B_1} \subseteq U_i$ and $\overline{B_2} \subseteq V_i$. Conversely, assume that $(A_1, B_1) \ll (A_2, B_2)$. There exist directed collections of open sets $(U_i)_{i\in I}$ and $(V_i)_{i\in I}$ with union A_2 and B_2 respectively such that $\overline{U_i}$ and $\overline{V_i}$ are compact subsets of A_2 and B_2 for each $i \in I$. By the definition of the way-below relation, there exists $i \in I$ with $A_1 \subseteq U_i$ and $B_1 \subseteq V_i$ from which it follows that $\overline{A_1}$ and $\overline{B_1}$ are compact subsets of A_2 and B_2 respectively. Every open subset of $\mathbb{R}^n$ can be obtained as the union of an increasing sequence of open rational polyhedra (i.e. polyhedra whose vertices have rational coordinates) way-below the open set. The collection of all pairs of disjoint open rational polyhedra thus provides a countable basis for $\mathbf{S}\mathbb{R}^n$. □

In practice, we are often interested in the subdomain $\mathbf{S}_b\mathbb{R}^n$ of *bounded* partial solids which is defined as $\mathbf{S}_b\mathbb{R}^n = \{(A, B) \in \mathbf{S}\mathbb{R}^n \mid B^c \text{ is bounded}\} \cup \{(\emptyset, \emptyset)\}$, ordered by inclusion. It is easy to see that $\mathbf{S}_b\mathbb{R}^n$ is a subdcpo of $\mathbf{S}\mathbb{R}^n$. Moreover, it is left as an exercise to show that:

Proposition 14.6. *The dcpo* $\mathbf{S}_b\mathbb{R}^n$ *is* ω*-continuous with the way-below relation given by* $(A_1, B_1) \ll (A_2, B_2)$ *iff* $\overline{A_1} \subseteq A_2$ *and* $B_2^c \subseteq {B_1}^{c\circ}$.

We say $(A, B) \in \mathbf{S}[-a, a]^n$ is a *proper* element if $(A, B) \neq (\emptyset, [-a, a]^n)$ and $(A, B) \neq ([-a, a]^n, \emptyset)$. Consider the collection $\mathcal{R}([-a, a]^n)$ of non-empty regular closed subsets of $[-a, a]^n$ with the metric given by,

$$d(A, B) = \max(d_H(A, B), d_H(\overline{A^c}, \overline{B^c}),$$

where d_H is the Hausdorff metric.

Theorem 14.7. *The collection of proper maximal elements of* $\mathbf{S}[-a,a]^n$ *is the continuous image of the space* $(\mathcal{R}([-a,a]^n), d)$ *of the non-empty regular closed subsets of* $[-a,a]^n$.

Proof. It is convenient to work with the representation of $\mathbf{S}[-a,a]^n$ by pairs (A,B) of closed subsets of $[-a,a]^n$, with $A \cup B = [-a,a]^n$, ordered by reverse inclusion. Any pair of open sets (U,V) of $[-a,a]^n$ provides a basic Scott open set $O_{(U,V)}$ of $\mathbf{S}[-a,a]^n$ given by $O_{(U,V)} = \{(A,B) \in \mathbf{S}[-a,a]^n \mid A \subset U \,\&\, B \subset V\}$. Now consider the map $\Gamma : \mathcal{R}([-a,a]^n) \to \mathbf{S}[-a,a]^n$ defined by $\Gamma(A) = (A, \overline{A^c})$. Clearly, Γ is a function onto the set of proper maximal elements of $\mathbf{S}[-a,a]^n$. To show that it is continuous, suppose $(A, \overline{A^c}) \in O_{(U,V)}$, i.e. $A \subset U$ and $\overline{A^c} \subset V$. Let $k = \min(r(A, U^c), r(\overline{A^c}, V^c))$ where $r(Y,Z)$ is the minimum distance between compact sets Y and Z. Then for $D \in \mathcal{R}([-a,a]^n)$ with $d(C,D) < k$, the inequalities $d_H(C,D) < k$ and $d_H(\overline{C^c}, \overline{D^c}) < k$ imply $D \subset U$ and $\overline{D^c} \subset V$. This shows that Γ is continuous. □

We can define a metric on the non-empty closed subsets of $\mathbb{R}^n$ by putting: $d'_H(A,B) = \max(d_H(A,B), 1)$. We leave it as an exercise for the reader to show that the collection of proper maximal elements of $\mathbf{S}\mathbb{R}^n$ is the continuous image of the space $(\mathcal{R}(\mathbb{R}^n), d')$ of the non-empty regular closed subsets of $\mathbb{R}^n$ with the metric defined by

$$d'(A,B) = \max(d'_H(A,B), d'_H(\overline{A^c}, \overline{B^c})). \tag{48}$$

15 Predicates and Operations on Solids

Our definition is also consistent with requirement 2 in a closely related way. We consider the idealisation of a machine used to measure mechanical parts. Two parts corresponding to equivalent subsets cannot be distinguished by such a machine. Moreover, partial solids, and, more generally, domain-theoretically defined data types allow us to capture partial, or uncertain input data encountered in realistic CAD situations. In order to be able to compute the continuous membership predicate, we extend it to the interval domain $\mathbf{I}\mathbb{R}^n$ and define $- \in - : \mathbf{I}\mathbb{R}^n \times \mathbf{S}\mathbb{R}^n \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$ with:

$$C \in (A,B) = \begin{cases} \mathsf{tt} & \text{if } C \subseteq A \\ \mathsf{ff} & \text{if } C \subseteq B \\ \perp & \text{otherwise} \end{cases}$$

(see Figure 2). Note that we use the infix notation for predicates and Boolean operations.

We define the predicate $- \subseteq - : \mathbf{S}_b\mathbb{R}^n \times \mathbf{S}\mathbb{R}^n \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$, by

$$(A,B) \subseteq (C,D) = \begin{cases} \mathsf{tt} & \text{if } B \cup C = \mathbb{R}^n \\ \mathsf{ff} & \text{if } A \cap D \neq \emptyset \\ \perp & \text{otherwise} \end{cases}$$

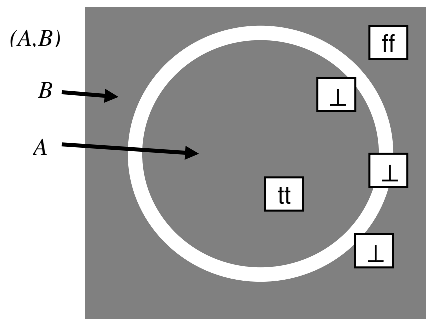

Fig. 2. The membership predicate of a partial solid object of the unit square.

The restriction to $\mathbf{S}_b\mathbb{R}^n$ will ensure that $- \subseteq -$ is continuous, as we will see in one of the exercises below. Starting with the continuous membership predicate, the natural definition for the complement would be to swap the values tt and ff. This means that the complement of (A, B) is (B, A), cf. requirement 3.

As for requirement 4, Figure 3 represents a subset S of $[0, 1]^2$ that is not regular. Its regularization removes both the external and internal "dangling edge". Here and in subsequent figures, the two components A and B of the partial solid are, for clarity, depicted separately below each picture.

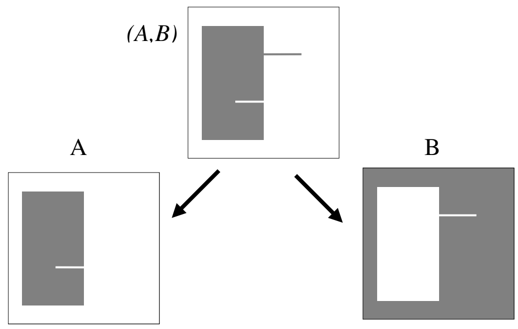

Fig. 3. Representation of a non-regular solid.

Bearing in mind that for a partial solid object (A, B), the open sets A and B respectively capture the interior and the exterior of the solid, we can deduce the definition of Boolean operators on partial solids:

$$(A_1, B_1) \cup (A_2, B_2) \;=\; (A_1 \cup A_2, B_1 \cap B_2)$$

$$(A_1, B_1) \cap (A_2, B_2) \;=\; (A_1 \cap A_2, B_1 \cup B_2).$$

One can likewise define the m-ary union and the m-ary intersection of partial solids. Note that, given two partial solids representing adjacent boxes, their union would not represent the set-theoretic union of the boxes, as illustrated in Figure 4.

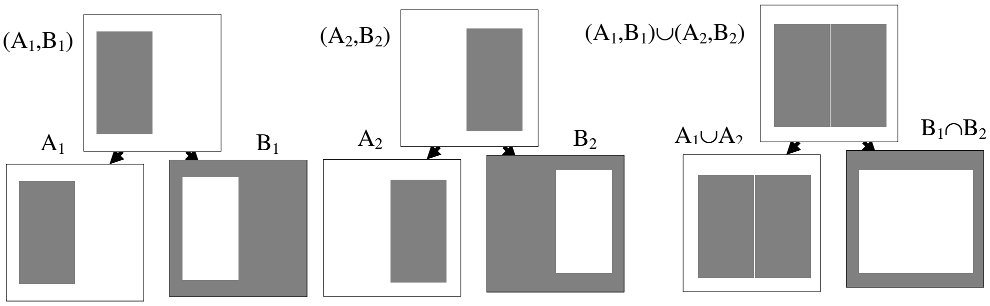

Fig. 4. The union operation on the solid domain.

Theorem 15.1. *The following maps are continuous:*

(i) *The predicate* $- \in - : \mathbf{IR}^n \times \mathbf{SR}^n \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$.

(ii) *The binary union* $- \cup - : \mathbf{SR}^n \times \mathbf{SR}^n \to \mathbf{SR}^n$ *and more generally the* m*-ary union* $\bigcup : (\mathbf{SR}^n)^m \to \mathbf{SR}^n$.

(iii) *The binary intersection* $- \cap - : \mathbf{SR}^n \times \mathbf{SR}^n \to \mathbf{SR}^n$ *and more generally the* m*-ary intersection* $\bigcap : (\mathbf{SR}^n)^m \to \mathbf{SR}^n$.

Proof. (i) A function of two variables on domains is continuous iff it is continuous in each variable separately when the other variable is fixed. From this, we obtain the required continuity by observing that a non-empty compact set is contained in the union of an increasing sequence of open sets iff it is contained in one such open set.

(ii) This follows from the distributivity of $\cup$ over $\cap$.

(iii) Follows from (ii) by duality. $\square$

15.1 The Minkowski Operator

We now introduce the Minkowski sum operation for partial solids of $\mathbb{R}^n$. Recall that the Minkowski sum of two subsets $S_1, S_2 \subseteq \mathbb{R}^n$ is defined as

$$S_1 \oplus S_2 = \{x + y \mid x \in S_1, y \in S_2\}$$

where $x + y$ is the vector addition in $\mathbb{R}^n$. For convenience we will use the same notation $\oplus$ for the Minkowski sum on the solid domain, which is defined as a function $- \oplus - : (\mathbf{S}_b\mathbb{R}^n) \times (\mathbf{S}\mathbb{R}^n) \to \mathbf{S}\mathbb{R}^n$ by:

$$(A_1, B_1) \oplus (A_2, B_2) = ((A_1 \oplus A_2), (B_1^c \oplus B_2^c)^c).$$

It can be shown that $- \oplus - : (\mathbf{S}_b\mathbb{R}^d) \times (\mathbf{S}\mathbb{R}^d) \to \mathbf{S}\mathbb{R}^d$ is well-defined and continuous.

16 Computability on the Solid Domain

We can provide an effective structure for $\mathbf{S}\mathbb{R}^n$ as follows. Consider the collection of all pairs of disjoint open rational polyhedra of the form $K = (L_1, L_2)$. Take an effective enumeration $(K_i)_{i\in\omega}$ with $K_i = (\pi_1(K_i), \pi_2(K_i))$ of this collection.

We say (A, B) is a *computable* partial solid if there exists a total recursive function $\beta : \mathbb{N} \to \mathbb{N}$ such that $(A, B) = (\bigcup_{n\in\omega} \pi_1(K_{\beta(n)}), \bigcup_{n\in\omega} \pi_2(K_{\beta(n)}))$.

One can similarly define an effective structure on $\mathbf{I}\mathbb{R}^n$, by taking an effective enumeration of rational intervals.

It follows from the general domain-theoretic definition (see the Appendix) that a function $F : (\mathbf{S}\mathbb{R}^n)^2 \to \mathbf{S}\mathbb{R}^n$ is *computable* if the relation $\{(i, j, k) \mid K_k \ll F(K_i, K_j)\}$ is r.e.. The definition extends in the natural way to functions of other types. A sequence $((A_n, B_n))_{n\in\omega}$ of partial solids is *computable* if there exists a total recursive function $\alpha : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $(A_n, B_n) = \bigsqcup_{i\in\omega} K_{\alpha(n,i)}$, with $(K_{\alpha(n,i)})_{i\in\omega}$ an increasing chain for each $n \in \omega$. For domains in general, it can be shown that a function is computable iff it sends computable sequences to computable sequences.

Proposition 16.1. *The following functions are computable with respect to the effective structures on* $\mathbf{I}\mathbb{R}^n$, $\mathbf{S}\mathbb{R}^n$ *and* $\mathbf{S}[-a, a]^n$.

(i) $- \in - : \mathbf{I}\mathbb{R}^n \times \mathbf{S}\mathbb{R}^n \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$.
(ii) $- \cup - : \mathbf{S}\mathbb{R}^n \times \mathbf{S}\mathbb{R}^n \to \mathbf{S}\mathbb{R}^n$.
(iii) $- \cap - : \mathbf{S}\mathbb{R}^n \times \mathbf{S}\mathbb{R}^n \to \mathbf{S}\mathbb{R}^n$.
(iv) $- \subseteq - : \mathbf{S}[-a, a]^n \times \mathbf{S}[-a, a]^n \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$.

Proof. We show (ii) and leave the rest as exercise. We have to show that the relation $K_k \ll K_i \cup K_j$ is r.e. Writing this relation in detail, it reduces to

$$(\pi_1(K_k), \pi_2(K_k)) \ll (\pi_1(K_i) \cup \pi_1(K_j), \pi_2(K_i) \cap \pi_2(K_j)),$$

i.e. $\overline{\pi_1(K_k)} \subseteq \pi_1(K_i) \cup \pi_1(K_j)$ and $\overline{\pi_2(K_k)} \subseteq \pi_2(K_i) \cap \pi_2(K_j)$, which are both decidable. □

17 Lebesgue and Hausdorff Computability

Our domain-theoretic notion of computability so far has the essential weakness of lacking a quantitative measure for the rate of convergence of basis elements to a computable element. This shortcoming can be redressed by enriching the domain-theoretic notion of computability with an additional requirement which allows a quantitative degree of approximation. We will see in this section that this can be done in at least two different ways. The reader should refer to the appendix for various notions of computability in this section.

17.1 Lebesgue Computability

The Lebesgue measure μ in $\mathbb{R}^n$, which measures the volume of subsets of $\mathbb{R}^n$, gives us a notion of approximation which is stable under Boolean operations. For simplicity, we confine ourselves to the solid domain of a large cube in $\mathbb{R}^n$. We say that $(A, B) \in \mathbf{S}[-a, a]^n$ is *Lebesgue computable* if there exists a total recursive function $\beta : \mathbb{N} \to \mathbb{N}$ such that $(A, B) = (\bigcup_{n\in\omega} \pi_1(K_{\beta(n)}), \bigcup_{n\in\omega} \pi_2(K_{\beta(n)}))$ with $\mu(A) - \mu(\pi_1(K_{\beta(n)})) < 2^{-n}$ and $\mu(B) - \mu(\pi_2(K_{\beta(n)})) < 2^{-n}$. The definition extends naturally to computable elements of $(\mathbf{S}X)^m$ for any positive integer m.

Proposition 17.1. *If a is a computable real number and $(A, B) \in \mathbf{S}[-a, a]^n$ is a computable maximal element with $\mu(\partial A) = 0$, then (A, B) is Lebesgue computable.*

The sequence $((A_n, B_n))_{n\in\omega}$ is said to be Lebesgue computable if it is computable and if $(\mu(A_n))_{n\in\omega}$ and $(\mu(B_n))_{n\in\omega}$ are computable sequences of real numbers. As for computable elements, the definition extends naturally to computable sequences of $(\mathbf{S}X)^m$ for any positive integer m.

A computable function $f : (\mathbf{S}X)^m \to \mathbf{S}X$ is said to be *Lebesgue computable* if it takes any Lebesgue computable sequence of m-tuples of partial solids to a Lebesgue computable sequence of partial solids. The main result here is the following.

Theorem 17.2. *Boolean operations are Lebesgue computable.*

17.2 Hausdorff Computability

Another appropriate form for the quantitative degree of approximation of solids is provided by the Hausdorff distance. We say $(A, B) \in \mathbf{S}[-a, a]^n$ is *Hausdorff computable* if there exists a total recursive function $\beta : \mathbb{N} \to \mathbb{N}$ such that $(A, B) = (\bigcup_{n\in\omega} \pi_1(K_{\beta(n)}), \bigcup_{n\in\omega} \pi_2(K_{\beta(n)}))$ with $d_H(\pi_1(K_{\beta(n)}), A) < 2^{-n}$ and $d_H(\pi_2(K_{\beta(n)}), B) < 2^{-n}$.

We can define the notion of a Hausdorff computable map similar to the way we defined a Lebesgue computable map. The Hausdorff distance provides a good way of approximating solids; in fact, objects with small Hausdorff distance with each other are visually close. However, it can be shown by a non-trivial example that the binary Boolean operations do not preserve Hausdorff computability. The main positive result is the following.

Theorem 17.3. *A computable maximal element of* $\mathbf{S}[-a,a]^n$ *is Hausdorff computable.*

18 The Convex Hull

We have already seen that points of $\mathbb{R}^n$ can be modelled using the domain $\mathbf{I}\mathbb{R}^n$ of the compact rectangles in $\mathbb{R}^n$ ordered by reverse inclusion. Using the domain-theoretic model, one can construct other basic notions in geometry, such as line segments, lines and hyperplanes. We demonstrate this by describing the simplest non-trivial geometric object, namely a line segment.

We define the *partial line segment map* $f : (\mathbf{I}\mathbb{R}^n)^2 \to \mathbf{S}\mathbb{R}^n$ with $f(x_1, x_2)$, called the *partial line segment* through the partial points x_1 and x_2, given by $f(x_1, x_2) = (\emptyset, E)$ where the exterior E is the empty set if $x_1 \cap x_2 \neq \emptyset$ and is otherwise the complement of the convex hull of the 2×2^n vertices of x_1 and x_2; see Figure 5. It is easy to check that f is Scott continuous and computable. Likewise, one can define Scott continuous maps for partial lines through two partial points, and other basic geometric objects.

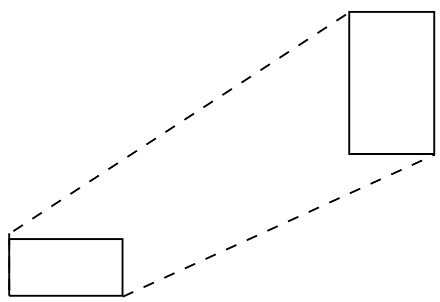

Fig. 5. A partial line segment

We will now describe an algorithm to compute the convex hull of a finite number of points in the plane in the context of the solid domain. Assume we have m points in the plane. Each of these points is approximated by a shrinking nested sequence of rational rectangles; at each finite stage of computation we have approximations to the m points by m rational rectangles, considered as imprecise points, as in Figure 6.

For these m rational rectangles we obtain a partial solid object with an interior open rational polygon, which is contained in the interior of the convex hull of the m points, and an exterior open rational polygon, which is contained in the exterior of the convex hull of the m points. The union of the interior (respectively, the exterior) open rational polygons obtained for all finite stages of computation gives the interior (respectively, the exterior) of the convex hull of the m points.

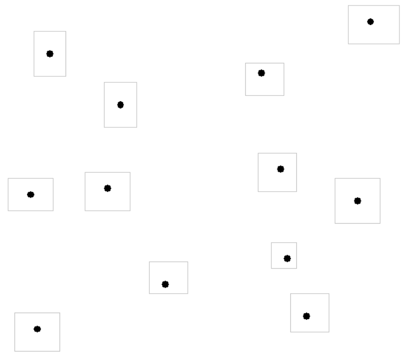

Fig. 6. The convex hull problem for rectangles.

More formally, we define a map $C_m : (\mathbf{I}\mathbb{R}^2)^m \to \mathbf{S}\mathbb{R}^2$, where $\mathbf{I}\mathbb{R}^2$ is the domain of the planar rectangles, the collection of all rectangles of the plane partially ordered by reverse inclusion. Let $\mathcal{C}(\mathbb{R}^2)$ be the collection of non-empty compact subsets of $\mathbb{R}^2$ with the Hausdorff metric and let $H_m : (\mathbb{R}^2)^m \to \mathcal{C}(\mathbb{R}^2)$ be the classical function which sends any m-tuple of planar points to its convex hull regarded as a compact subset of the plane.

We first define C_m on the basis $(\mathbf{I}\mathbb{Q}^2)^m$ of $(\mathbf{I}\mathbb{R}^2)^m$ consisting of m-tuples of rational rectangles. Let $x = (R_1, R_2, \cdots, R_m) \in (\mathbf{I}\mathbb{Q}^2)^m$ be an m-tuple of rational rectangles. Each rectangle R_i has four vertices denoted, anti-clockwise starting with the bottom left corner, by R_i^1, R_i^2, R_i^3 and R_i^4. We define $C_m(x) = (I_m(x), E_m(x))$ with

$$E_m(x) = (H_{4m}((R_i^1, R_i^2, R_i^3, R_i^4))_{i=1}^m)^c, \qquad I_m(x) = (\bigcap_{1 \leq j \leq 4} H_m(R_i^j)_{i=1}^m)^\circ.$$

In words, $E_m(x)$ is the complement of the convex hull of the $4m$ vertices of all rectangles (Figure 7), whereas $I_m(x)$ is the interior of the intersection of the 4 convex hulls of the bottom left, bottom right, top right and top left vertices (Figure 8). Since the intersection of convex sets is convex, $I_m(x)$ as well as $E_m(x)$ are both convex open rational polygons.

With more accurate input data about the planar points, the boundaries of the inner and outer convex hulls get closer to each other as in Figure 9. In the limit, the inner and outer convex hulls will be simply the interior and the exterior of the convex hull of the planar points (Figure 10).

Since we work completely with rational arithmetic, we will not encounter any round-off errors and, as comparison of rational numbers is decidable, we will not get inconsistencies.

Clearly the complexity of these algorithms to compute $I_m(x)$ and $E_m(x)$ is $O(m \log m)$ each. We have therefore obtained a robust algorithm for the convex

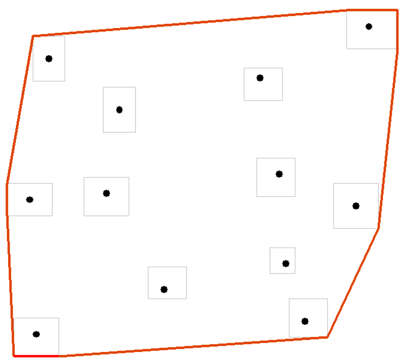

Fig. 7. The exterior convex hull of rectangles.

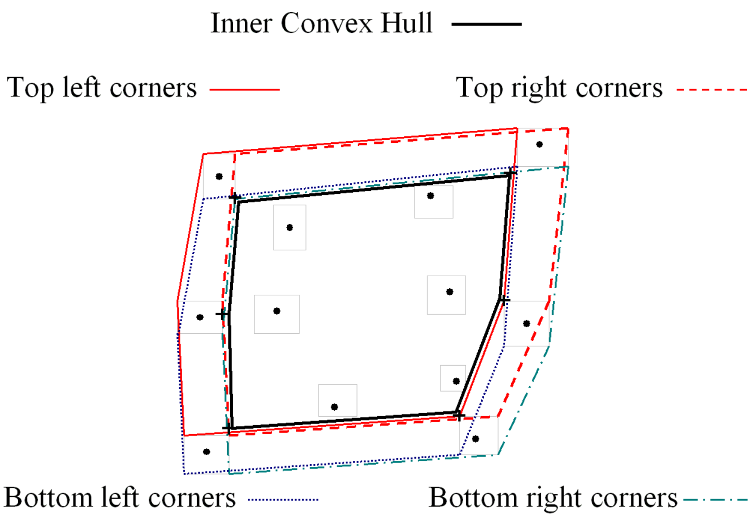

Fig. 8. The interior convex hull of rectangles.

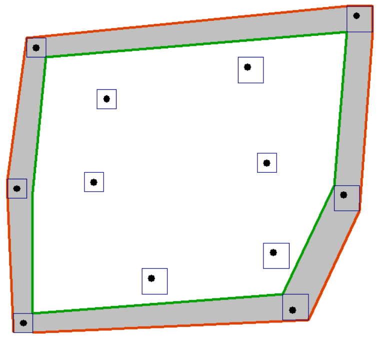

Fig. 9. Convergence of the interior and exterior convex hulls.

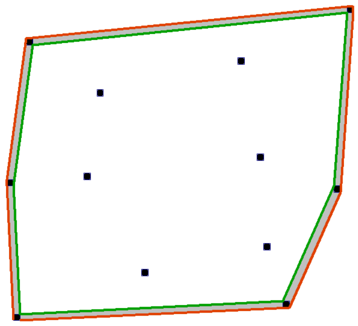

Fig. 10. Limit of interior and exterior convex hulls.

hull which has the same complexity as the non-robust classical algorithm. Moreover, the algorithm extends in the obvious way to $\mathbb{R}^d$. In 3d, we still have the complexity $O(m \log m)$; see [15] for the complexity in higher dimension.

We now define $\hat{C}_m : (\mathbf{I}\mathbb{R}^2)^m \to \mathbf{S}\mathbb{R}^2$ on tuples of rectangles $y \in (\mathbf{I}\mathbb{R}^2)^m$ by putting $\hat{C}_m(y) = \bigsqcup\{C(x) \mid x \in (\mathbf{I}\mathbb{Q})^m \text{ with } x \ll y\}$. It can be checked that $\hat{C}_m(x) = C_m(x)$ for $x \in (\mathbf{I}\mathbb{Q}^2)^m$, and that, therefore, we can simply write $\hat{C}_m$ as C_m which will be a continuous function between domains.

The map C_m computes the convex hull of m planar points as follows. Note that a maximal element $x = (R_i)_{i=1}^m$ of $(\mathbf{I}\mathbb{R}^2)^m$ consists of an m-tuple of degenerate rectangles, i.e., an m-tuple of planar points $(r_i)_{i=1}^m$, where $R_i^j = r_i$, for $j = 1, 2, 3, 4$. It can be shown that, for such maximal x, we have $C_m(x) = (I_m(x), E_m(x))$ where $I_m(x) = (H_m((r_i)_{i=1}^m))^\circ$ and $E_m(x) = (H_m((r_i)_{i=1}^m))^c$.

Theorem 18.1. *The map C_m is Lebesgue computable and Hausdorff computable.*

We can also study the domain-theoretic version of the following classical question: Given N points $x_1, \ldots, x_N$ in $\mathbb{R}^2$, does x_k, for $1 \leq k \leq N$, lie on the boundary of the convex hull of these N points? With imprecise input, i.e. for N input rectangles, the answer is either "surely yes", or "surely not" or "cannot say". More precisely, we define the *boundary rectangle* predicate $P_k : (\mathbf{I}\mathbb{R}^2)^N \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$. For $\overline{R} = (R_1, \ldots, R_N) \in (\mathbf{I}\mathbb{R}^2)^N$, let $\overline{R}(k) \in (\mathbf{I}\mathbb{R}^2)^{N-1}$ be the ordered list of the $N-1$ dyadic interval vertices: $\overline{R}(k) = (R_1, \ldots, R_{k-1}, R_{k+1}, \ldots, R_N)$. We have:

$$P_k(\overline{R}) = \begin{cases} \mathsf{tt} \text{ if } R_k \subseteq E(\overline{R}(k)), \\ \mathsf{ff} \text{ if } R_k \subseteq I(\overline{R}), \\ \perp \text{ otherwise.} \end{cases} \tag{49}$$

Theorem 18.2. *The predicate P_k is Scott continuous and computable for each $k = 1, \ldots, N$.*

Finally, we note that domain-theoretic algorithms for Voronoi diagram and Delaunay triangulation have also been developed; see [38].

19 Historical Remarks and Pointers to Literature

19.1 Real Number Computation

In the late 1980's, two frameworks for exact real number computation were proposed. In the approach of Boehm and Cartwright [5, 6], a computable real number is approximated by rational numbers of the form K/r^n where r is the base and K is a (usually big) integer. This approach was further developed and implemented by Valérie Ménissier-Morain [40]. For any basic function in analysis a feasible algorithm has been presented in order to produce an approximation to the value of the function at a given computable real number up to any threshold of accuracy. However, the computation is not incremental in the sense that to obtain a more accurate approximation one has to compute from scratch. Furthermore, the algorithms are constructed using various ad-hoc techniques and therefore, except for the simplest arithmetic operations, it is rather difficult to verify their correctness. Actually, this method is not so different from the multi-digit approach presented here, except that our transcendental operations are based on LFT's, which provide a general underlying framework that simplifies the finding of the algorithms and makes the proofs of their correctness automatic.

Vuillemin [57] proposed a representation of computable real numbers by continued fractions and presented various incremental algorithms for basic arithmetic operations using the earlier work of Gosper [24], and for some transcendental functions. However, this representation is rather complicated and the resulting algorithms are relatively inefficient.

Plume [42] studied and implemented Exact Real Arithmetic based on the number representation of Section 2 (exponent plus a stream of signed binary digits). His division algorithm employs an auxiliary representation with dyadic rationals as digits. Transcendental functions are based on an auxiliary function computing the real number defined by a (computable) nested sequence of real intervals whose lengths tend to 0.

In the early and mid 90's Di Gianantonio [12, 13] and Escardó [20] studied extensions of the theoretical language PCF with a real number data type based on domain theory. At Imperial College, a new approach was then developed which is almost entirely based on LFT's and combines domain theory and the digit approach with continued fraction algorithms [45, 16, 46, 43, 44]. Within this approach, Peter Potts derived algorithms for transcendental functions from continued fraction expansions. He also developed the single-digit approach with the absorption and emission of digit matrices, and made first steps towards a multi-digit approach. The approach was implemented in functional languages such as Miranda, Haskell and CAML, and in imperative languages such as C. The LFT framework for real number computation has also been studied in the context of extensions of PCF with a real number data type by Edalat, Escardó, Potts and Sünderhauf [47, 17].

In contrast to the notes at hand, Potts and Edalat used the base interval $[0, \infty]$, and accordingly, digit matrices which were different from the ones presented here. This approach includes ∞ with the same rights as any finite real number. The number ∞ represents both $+\infty$ and $-\infty$. Its presence makes the reciprocal function total by $\frac{1}{0} = \infty$ and $\frac{1}{\infty} = 0$. Yet on the other hand, addition and multiplication, which are total if ∞ is excluded, become partial with ∞ since $\infty + \infty$ and $0 \cdot \infty$ are not defined.

In this approach, exponent matrices cannot be used. Instead, each number representation begins with a *sign matrix*. There are four sign matrices, for numbers in the intervals $[0, \infty]$, $[-1, 1]$, $[\infty, 0]$, and $[1, -1] = \{x \| x| \geq 1\}$. Edalat and Potts name two advantages of $[0, \infty]$: First, the image $M[0, \infty]$ of $[0, \infty]$ under a non-singular LFT $M = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ can be easily obtained from the entries of M: $M[0, \infty] = [\frac{c}{d}, \frac{a}{b}]$ if $\det M > 0$, and $[\frac{a}{b}, \frac{c}{d}]$ if $\det M < 0$. In contrast, the calculation of $M[-1, 1]$ requires some additions. Second, a matrix or tensor is refining for $[0, \infty]$ iff all its entries are non-negative and all its column sums are positive (if the matrix or tensor is weakly normalised so that the sum of its entries is non-negative). This condition is much simpler than the conditions we have derived for refinement w.r.t. $[-1, 1]$ in Section 5. The emission conditions for the two base intervals are similar, but the actual emissions and absorptions are simpler in $[-1, 1]$. A huge practical advantage of $[-1, 1]$ are the persistent zeros which can be found in basically all the tensors for the standard operations. With $[0, \infty]$, there are no persistent zeros at all, and no entries which are invariant under absorption and emission.

On the theoretical side as well, the base interval $[-1, 1]$ has clear advantages. It avoids the troublesome value ∞ that poses difficulties in algebraic transformations and size estimations. Furthermore, one may work with the standard metric $(\ell([u, v]) = v - u)$ and standard derivatives in $[-1, 1]$, while working with $[0, \infty]$ excludes the standard metric. In fact, [16, 43, 28] use a metric on $[0, \infty]$ that is derived from the standard metric on $[-1, 1]$. Here, working in $[-1, 1]$ directly drastically simplifies the reasoning.

Results on the growth of the entries of matrices and tensors were presented in [26, 27]—for the base interval $[0, \infty]$. With this base interval, matrices $\left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ cannot be classified according to $b = 0$ and $b \neq 0$ as in Section 8; the crucial value is instead $(c + d) - (a + b)$. Given this, it is not surprising that a complete classification of tensors w.r.t. the opportunities for cancellations was never found under the reign of $[0, \infty]$. The classification presented in Section 8.5 of these notes was recently found by Reinhold Heckmann and never published before.

The contractivity was already studied by Potts, and considered in greater detail by Heckmann in [28] (for $[0, \infty]$). In [30], Heckmann switched over to $[-1, 1]$ and studied contractivity there.

Peter Potts was a master in the derivation of infinite products from continued fractions (for $[0, \infty]$). The few derivations presented here are new because of the new base interval. They start from the same continued fractions, but are generally shorter. The derivation of products from Taylor series is taken from [29].

19.2 Computational Geometry

The quest for reliable geometric algorithms in recent years has been a most challenging problem. In the words of C. M. Hoffmann, a leading expert in computational geometry: "Despite the pressing need, devising accurate and robust geometric algorithms has proved elusive for many important science and engineering applications" [31].

In the existing frameworks and implementations of geometric algorithms, great efforts are required to use various, often ad hoc, techniques in order to avoid potential inconsistencies and degeneracies. These methods include: (i) the so-called exact arithmetic approach [41, 37, 51, 23, 52, 3, 61, 9, 8, 22, 11], combined with lazy implementation [4, 53] and symbolic perturbation [19, 51, 60] in which numerical computations are performed to a high degree of accuracy in order to ensure the correct logical and topological decisions; (ii) the logical and topological oriented technique [52, 55, 56], which seeks to place the highest priority on the consistency of the logical and topological properties of geometric algorithms, using numerical results only when they are consistent with these properties; and, (iii) the intermediate methods, such as ϵ-geometry [25], the interval arithmetic technique [49, 32–34] and the tolerance approach [50, 21, 36], which determine an upper bound for the numerical error whenever a computation takes place in order to decide if a computation is reliable or not. While there are pros and cons for each of these methods in any given category of algorithms [54]; no single method gives an overall satisfactory solution for geometric modelling as a whole.

The traditional frameworks for geometric modelling are not founded on computable analysis: there is no reference to a notion of data type or computability in the standard literature of computational geometry or geometric modelling. Indeed, these frameworks are all based on classical topology and geometry in which the basic predicates and Boolean operations are not continuous, and hence not computable, the source of non-robustness of the resulting algorithms.

Brattka and Weihrauch [7] have studied the question of computability of classical subsets of the Euclidean space in the type two theory of computability [59] but it is not at all clear how their framework can be used in any practical geometric computation.

The domain-theoretic framework for solid modelling and computational geometry was first formulated in [14] and algorithms for the convex hull and for Voronoi diagram/Delaunay triangulation in the domain-theoretic setting were presented in [15] and [38] respectively. Continuous geometric operations have also been discussed in [35].

20 Exercises

20.1 Real Arithmetic

Exercise 20.1. Implement addition $x+y$ directly on the number representations by exponents and signed binary digit streams (cf. Section 2.4). Deal first with exponents and use the mean value operation $x \oplus y = \frac{x+y}{2}$ on mantissas.

Exercise 20.2. Prove Prop. 3.1 (using Equation (1)).

Exercise 20.3. Let $M_0 = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 3 \end{smallmatrix}\right)$.

a) What is the function represented by M_0?
b) Compute $\det M_0$ (Equation (2)) and deduce the monotonicity type of M_0 (Section 4).
c) Check that M_0 is bounded (Prop. 5.2) and refining (Section 5.3) on I_0.
d) Assuming that the digit stream ξ starts with **101**, determine the first *four* digits of $M_0(\xi)$ as in Section 6.5.
e) Compute $\exp M_0$ and $\operatorname{con} M_0$ (23) and derive the numbers $c^<$ and $c^>$ of Theorem 7.1.
f) Redo part (d) in the multi-digit approach, i.e., answer the request $4?\,M_0(\xi)$. Run Algorithm 3, but consider the monotonicity type of M_0. Use the fact that ξ begins with **101** to find the answer of the required request to ξ.
g) Compare the results of parts (d) and (f), but remember that there are often two possible answers to a request, differing by 1.

Exercise 20.4. Let $T = \left(\begin{smallmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 3 \end{smallmatrix}\right)$.

a) What is the function represented by T?
b) Compute $\det(T|_x)$ and $\det(T|^y)$ (Equations (6)) and deduce the monotonicity type of T for arguments $x, y \in I_0$ (Section 4).
c) Check that T is bounded (14) and refining on I_0. For the latter, you may use (15) or Cor. 4.2, taking the monotonicity type into account.
d) Determine $\operatorname{con}_{\mathrm{L}} T$ and $\operatorname{con}_{\mathrm{R}} T$ and derive the numbers $c_{\mathrm{L}}^{\gtrless}$ and $c_{\mathrm{R}}^{\gtrless}$ of Theorem 7.2.

Exercise 20.5. Let $T = \left(\begin{smallmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{smallmatrix}\right)$. Given $x \geq 0$, solve the equation $y = T(x, y)$ for $y \geq 0$. (Thus you see how an important function can be implemented. The equation $y = T(x, y)$ can be considered as an infinite product $y = T(x, T(x, \ldots))$, or more efficiently, as a feed-back loop where everything emitted from T is fed back into T via its right argument.)

Exercise 20.6. (Taylor series)
Use the method presented in Section 11.4 to derive an infinite product for the cosine function from the Taylor series $\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!}(x^2)^n$. (By writing this in terms of x^2 instead of x, zero coefficients are avoided.) Determine for which x this product is valid, and calculate the contractivities of its matrices.

20.2 Computational Geometry and Solid Modelling

Exercise 20.7. Show that the map $- \subseteq - : \mathbf{S}_b\mathrm{I\!R}^n \times \mathbf{S}\mathrm{I\!R}^n \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$ is continuous.

Exercise 20.8. Prove Proposition 14.6.

Hint: Use the following fact for Euclidean spaces. For an open set O and a decreasing sequence of compact subsets $\langle C_i \rangle_{i\omega}$, the relation $\bigcap_{i\in\omega} C_i \subseteq O$ implies the existence of $i \in \omega$ with $C_i \subseteq O$.

Exercise 20.9. Show that the collection of proper maximal elements of $\mathbf{S}\mathbb{R}^n$ is the continuous image of the space $(\mathcal{R}(\mathbb{R}^n), d')$ of the non-empty regular closed subsets of $\mathbb{R}^n$ with the metric defined by Equation 48.

Hint: Follow the steps of proof in Theorem 14.7 and note that in the representation of $\mathbf{S}\mathbb{R}^n$ by closed sets ordered by reverse inclusion we have: $(A_1, B_1) \ll (A_2, B_2)$ iff A_2 and B_2 are compact subsets of A_1° and B_1° respectively.

Exercise 20.10. Draw the inner and outer convex hulls of the following three rectangles.

$$\begin{aligned} R_1 &= \{(-2,0),(-1,0),(-1,-1),(-2,-1)\} \\ R_2 &= \{(-1,3),(0,3),(0,2),(-1,2)\} \\ R_3 &= \{(1,1),(2,1),(2,0),(1,0)\}. \end{aligned}$$

Exercise 20.11. In the domain-theoretic convex hull algorithm, compute the boundary rectangle predicate P_k for $1 \leq k \leq 11$.

$$\begin{aligned} R_1 &= \{(-7/2,-3),(-7/2,-2),(-5/2,-2),(-5/2,-3)\} \\ R_2 &= \{(-7/2,-1),(-7/2,-1/2),(-3,-1/2),(-3,-1)\} \\ R_3 &= \{(-4,4/3),(-4,5/3),(-3,5/3),(-3,4/3)\} \\ R_4 &= \{(-2,-4),(-2,-7/2),(-3/2,-7/2),(-3/2,-4)\} \\ R_5 &= \{(-2,3),(-2,7/2),(-3/2,7/2),(-3/2,3)\} \\ R_6 &= \{(0,-4),(0,-7/2),(1/2,-7/2),(1/2,-4)\} \\ R_7 &= \{(0,0),(0,1),(1,1),(1,0)\} \\ R_8 &= \{(0,4),(0,5),(1,5),(1,4)\} \\ R_9 &= \{(4,-3),(4,-2),(5,-2),(5,-3)\} \\ R_{10} &= \{(5,-1),(5,-1/2),(27/5,-1/2),(27/5,-1)\} \\ R_{11} &= \{(5,2),(5,3),(6,3),(6,2)\}. \end{aligned}$$

Hint: Note that a rectangle is a boundary rectangle if it lies completely inside the exterior convex hull of the other rectangles.

Appendix: Basic Domain Theory

We give here the formal definitions of a number of notions in domain theory used in these notes; see [1, 2, 18] for more detail. We think of a partially ordered set (poset) $(P, \sqsubseteq)$ as the set of output of some computation such that the partial order is an order of information: in other words, $a \sqsubseteq b$ indicates that a has less information than b. For example, the set $\{0,1\}^\infty$ of all finite and infinite sequences of bits 0 and 1 with $a \sqsubseteq b$ if the sequence a is an initial segment of the sequence b is a poset and $a \sqsubseteq b$ simply means that b has more bits of information than a. A non-empty subset $A \subseteq P$ is *directed* if for any pair of elements $a, b \in A$ there exists $c \in A$ such that $a \sqsubseteq c$ and $b \sqsubseteq c$. A directed set is therefore a consistent set of output elements of a computation: for every pair of output a and b, there is some output c with more information than a and b. A *directed complete partial order (dcpo)* or a *domain* is a partial order in which every directed subset has a least upper bound (lub). We say that a dcpo is *pointed* if it has a least element which is denoted by $\perp$ and is called *bottom*.

For two elements a and b of a dcpo we say a is *way-below* or *approximates* b, denoted by $a \ll b$, if for every directed subset A with $b \sqsubseteq \bigsqcup A$ there exists $c \in A$ with $a \sqsubseteq c$. The idea is that a is a finitary approximation to b: whenever the lub of a consistent set of output elements has more information than b, then already one of the input elements in the consistent set has more information than a. In $\{0,1\}^\infty$, we have $a \ll b$ iff $a \sqsubseteq b$ and a is a finite sequence. The closed subsets of the *Scott topology* of a domain are those subsets C which are downward closed (i.e. $x \in C \;\&\; y \sqsubseteq x \;\Rightarrow\; y \in C$) and closed under taking lub's of directed subsets (i.e. for every directed subset $A \subseteq C$ we have $\bigsqcup A \in C$).

A basis of a domain D is a subset $B \subseteq D$ such that for every element $x \in D$ of the domain the set $B_x = \{y \in B \mid y \ll x\}$ of elements in the basis way-below x is directed with $x = \bigsqcup B_x$. An *(ω)-continuous* domain is a dcpo with a (countable) basis. In other words, every element of a continuous domain can be expressed as the lub of the directed set of basis elements which approximate it. In a continuous dcpo D, subsets of the form $\twoheaduparrow a = \{x \in D \mid a \ll x\}$, for $a \in D$, form a basis for the Scott topology. A domain is *bounded complete* if every bounded subset has a lub; in such a domain every non-empty subset has an infimum or greatest lower bound.

It can be shown that a function $f : D \to E$ between dcpo's is continuous with respect to the Scott topology if and only if it is *monotone* (i.e. $a \sqsubseteq b \;\Rightarrow\; f(a) \sqsubseteq f(b)$) and preserves lub's of directed sets i.e. for any directed $A \subseteq D$, we have $f(\bigsqcup_{a\in A} a) = \bigsqcup_{a \in A} f(a)$. Moreover, if D is an ω-continuous dcpo, then f is continuous iff it is monotone and preserves lub's of increasing sequences (i.e. $f(\bigsqcup_{i\in\omega} x_i) = \bigsqcup_{i\in\omega} f(x_i)$, for any increasing $(x_i)_{i\in\omega}$).

The collection, $D \to E$, of continuous functions $f : D \to E$ between dcpo's D and E can be ordered pointwise: $f \sqsubseteq g$ iff $\forall x \in D.\, f(x) \sqsubseteq g(x)$. With this partial order, $D \to E$ becomes a dcpo with $\bigsqcup_{i\in I} f_i$ given by $(\bigsqcup_{i\in I} f_i)(x) = \bigsqcup_{i\in I} f_i(x)$. Moreover, if D and E are bounded complete ω-continuous dcpo's, so is $D \to E$.

The *interval domain* $\mathbf{I}[0,1]^n$ of the unit box $[0,1]^n \subseteq \mathbb{R}^n$ is the set of all non-empty n-dimensional sub-rectangles in $[0,1]^n$ ordered by reverse inclusion. A basic Scott open set is given, for every open subset O of $\mathbb{R}^n$, by the collection of all rectangles contained in O. The map $x \mapsto \{x\} : [0,1]^n \to \mathbf{I}[0,1]^n$ is an embedding onto the set of maximal elements of $\mathbf{I}[0,1]^n$. Every maximal element $\{x\}$ can be obtained as the least upper bound (lub) of an increasing chain of elements, i.e. a shrinking, nested sequence of sub-rectangles, each containing $\{x\}$ in its interior and thereby giving an approximation to $\{x\}$ or equivalently to x. The set of sub-rectangles with rational coordinates provides a countable basis. One can similarly define, for example, the interval domain $\mathbf{I}\mathbb{R}^n$ of $\mathbb{R}^n$.

An important feature of domains, in the context of these notes, is that they can be used to obtain computable approximations to operations which are classically non-computable. For example, comparison of a real number with 0 is not computable. However, the function $N : \mathbf{I}[-1,1] \to \{\mathsf{tt}, \mathsf{ff}\}_\perp$ with

$$N([a,b]) = \begin{cases} \mathsf{tt} & \text{if } b < 0 \\ \mathsf{ff} & \text{if } 0 < a \\ \perp & \text{otherwise} \end{cases}$$

is the computable approximation to the comparison predicate. Here, $\{\mathsf{tt}, \mathsf{ff}\}_\perp$ is the *lift* of $\{\mathsf{tt}, \mathsf{ff}\}$, i.e. the three element pointed domain with two incomparable maximal elements tt and ff.

An ω-continuous domain D with a least element $\perp$ is *effectively given* wrt an effective enumeration $b : \mathbb{N} \to B$ of a countable basis B if the set $\{\langle m, n\rangle \mid b_m \ll b_n\}$ is recursive, where $\langle .,. \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is the standard pairing function i.e. the isomorphism $(x,y) \mapsto \frac{(x+y)(x+y+1)}{2} + x$. This means that for each pair of basis elements (b_m, b_n), it is possible to decide in finite time whether or not $b_m \ll b_n$. We say $x \in D$ is *computable* if the set $\{n | b_n \ll x\}$ is r.e. This is equivalent to say that there is a master program which outputs exactly this set. It is also equivalent to the existence of a recursive function g such that $(b_{g(n)})_{n\in\omega}$ is an increasing chain in D with $x = \bigsqcup_{n\in\omega} b_{g(n)}$. If D is also effectively given wrt to another basis $B' = \{b'_0, b'_1, b'_2, \cdots\}$ such that the sets $\{\langle m,n\rangle \mid b_m \ll b'_n\}$ and $\{\langle m,n\rangle \mid b'_m \ll b_n\}$ are both decidable, then x will be computable wrt B iff it is computable wrt B'. We say that B and B' are *recursively equivalent*.

We can define an effective enumeration ξ of the set D_c of all computable elements of D. Let θ_n, $n \in \omega$, be the nth partial recursive function. It can be shown [18] that there exists a total recursive function σ such that $\xi : \mathbb{N} \to D_c$ with $\xi_n := \bigsqcup_{i\in\omega} b_{\theta_{\sigma(n)}(i)}$, with $(b_{\theta_{\sigma(n)}(i)})_{i\in\omega}$ an increasing chain for each $n \in \omega$, is an effective enumeration of D_c. A sequence $(x_i)_{i\in\omega}$ is *computable* if there exists a total recursive function h such that $x_i = \xi_{h(i)}$ for all $i \in \omega$.

We say that a continuous map $f : D \to E$ of effectively given ω-continuous domains D (with basis $\{a_0, a_1 \cdots\}$) and E (with basis $\{b_0, b_1 \cdots\}$) is *computable* if the set $\{\langle m,n\rangle \mid b_m \ll f(a_n)\}$ is r.e. This is equivalent to say that f maps computable sequences to computable sequences. Computable functions are stable under change to a recursively equivalent basis. Every computable function can be shown to be a continuous function [58, Theorem 3.6.16]. It can be shown [18] that these notions of computability for the domain $\mathbf{I}\mathbb{R}$ of intervals of $\mathbb{R}$ induce the same class of computable real numbers and computable real functions as in the classical theory [48].

We also need the following classical definitions for sequences of real numbers. A sequence $(r_i)_{i\in\omega}$ of rational numbers is *computable* if there exist three total recursive functions a, b, and s such that $b(i) \neq 0$ for all $i \in \omega$ and

$$r_i = (-1)^{s(i)} \frac{a(i)}{b(i)}.$$

A computable double sequence of rational numbers is defined in a similar way. A sequence $(x_i)_{i\in\omega}$ of real numbers is *computable* if there exists a computable double sequence $(r_{ij})_{i,j\in\omega}$ of rational numbers such that

$$|r_{ij} - x_i| \leq 2^{-j} \qquad \text{for all } i \text{ and } j$$

A computable double sequence of real numbers is defined analogously. If $(x_{nk})_{n,k\in\omega}$ is a computable double sequence of real numbers which converges to a sequence $(x_n)_{n\in\omega}$ effectively in k and n (i.e. there exists a total recursive function $e : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $|x_{nk} - x_n| \leq 2^{-N}$ for all $k \geq e(n, N)$), then the sequence $(x_n)_{n\in\omega}$ is computable [48, Page 20].

References

1. S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3. Clarendon Press, 1994.
2. R. M. Amadio and P.-L. Curien. *Domains and Lambda-Calculi.* Cambridge Tracts in Theoretical Computer Science, 1998.
3. F. Avnaim, J. D. Boissonnat, O. Devillers, F. Preparata, and M. Yvinec. Evaluation of a new method to compute signs of determinants. In *Proc. Eleventh ACM Symposium on Computational Geometry*, June 1995.
4. M. Benouamer, D. Michelucci, and B. Peroche. Error-free boundary evaluation using lazy rational arithmetic - a detailed implementation. In *Proceeding of the 2nd Symposium on Solid Modeling and Applications*, pages 115–126, 1993.
5. H. J. Boehm and R. Cartwright. Exact real arithmetic: Formulating real numbers as functions. In Turner. D., editor, *Research Topics in Functional Programming*, pages 43–64. Addison-Wesley, 1990.
6. H. J. Boehm, R. Cartwright, M. Riggle, and M. J. O'Donnell. Exact real arithmetic: A case study in higher order programming. In *ACM Symposium on Lisp and Functional Programming*, 1986.
7. V. Brattka and K. Weihrauch. Computability on subsets of Euclidean space I: Closed and compact subsets. *Theoretical Computer Science*, 219:65–93, 1999.
8. H. Brönimann, J. Emiris, V. Pan, and S. Pion. Computing exact geometric predicates using modular arithmetic with single precision. *ACM Conference on Computational Geometry*, 1997.
9. H. Brönnimann and M. Yvinec. Efficient exact evaluation of signs of determinants. In *Proc. Thirteenth ACM Symposium on Computational Geometry*, pages 136–173, June 1997.
10. L. E. J. Brouwer. Besitzt jede reelle zahl eine dezimalbruchentwicklung? *Math Ann*, 83:201–210, 1920.
11. O. Devillers, A. Fronville, B. Mourrain, and M. Teillaud. Algebraic methods and arithmetic filtering for exact predicates on circle arcs. In *Proc. Sixteenth ACM Symposium on Computational Geometry*, pages 139–147, June 2000.
12. P. Di Gianantonio. *A functional approach to real number computation.* PhD thesis, University of Pisa, 1993.
13. P. Di Gianantonio. Real number computability and domain theory. *Information and Computation*, 127(1):11–25, May 1996.
14. A. Edalat and A. Lieutier. Foundation of a computable solid modeling. In *Proceedings of the fifth symposium on Solid modeling and applications*, ACM Symposium on Solid Modeling and Applications, pages 278–284, 1999. Full paper to appear in TCS.

15. A. Edalat, A. Lieutier, and E. Kashefi. The convex hull in a new model of computation. In *Proc. 13th Canad. Conf. Comput. Geom.*, pages 93–96, 2001.
16. A. Edalat and P. J. Potts. A new representation for exact real numbers. In *Proceedings of Mathematical Foundations of Programming Semantics 13*, volume 6 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science B. V., 1997. Available from URL: http://www.elsevier.nl/locate/entcs/volume6.html.
17. A. Edalat, P. J. Potts, and P. Sünderhauf. Lazy computation with exact real numbers. In *Proceedings of the Third ACM SIGPLAN International Confrence on Functional Programming*, pages 185–194. ACM, 1998.
18. A. Edalat and P. Sünderhauf. A domain theoretic approach to computability on the real line. *Theoretical Computer Science*, 210:73–98, 1998.
19. H. Edelsbrunner and E. P. Mucke. Simulation of simplicity - a technique to cope with degenerate cases in geometric algorithms. In *Proceeding of the 4th ACM Annual Symposium on Computational Geometry*, pages 118–133, 1998.
20. M. H. Escardó. PCF extended with real numbers. *Theoretical Computer Science*, 162(1):79–115, August 1996.
21. S. Fang, B. Bruderlin, and X. Zhu. Robustness in solid modeling: a tolerance-based intuitionistic approach. *Computer-Aided Design*, 25(9):567–577, 1993.
22. S. Fortune. Polyhedral modeling with multi-precision integer arithmetic. *Computer-Aided Design*, 29(2):123–133, 1997.
23. S. Fortune and C. von Wyk. Efficient exact arithmetic for computational geometry. In *Proceeding of the 9th ACM Annual Symposium on Computational Geometry*, pages 163–172, 1993.
24. W. Gosper. *Continued Fraction Arithmetic.* HAKMEM Item 101B, MIT Artificial Intelligence Memo 239. MIT, 1972.
25. L. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry - building robust algorithms for imprecise computations. In *Proceeding of the 5th ACM Annual Symposium on Computational Geometry*, pages 208–217, 1989.
26. R. Heckmann. The appearance of big integers in exact real arithmetic based on linear fractional transformations. In *Proc. Foundations of Software Science and Computation Structures (FoSSaCS '98)*, volume 1378 of *LNCS*, pages 172–188. Springer-Verlag, 1998.
27. R. Heckmann. Big integers and complexity issues in exact real arithmetic. In *Third Comprox workshop (Sept. 1997 in Birmingham)*, volume 13 of *Electronic Notes in Theoretical Computer Science*, 1998. URL: `http://www.elsevier.nl/locate/entcs/volume13.html`.
28. R. Heckmann. Contractivity of linear fractional transformations. In J.-M. Chesneaux, F. Jézéquel, J.-L. Lamotte, and J. Vignes, editors, *Third Real Numbers and Computers Conference (RNC3)*, pages 45–59, April 1998. An updated version will appear in TCS.
29. R. Heckmann. Translation of Taylor series into LFT expansions. Submitted to Proceedings of Dagstuhl Seminar "Symbolic Algebraic Methods and Verification Methods", November 1999.
30. R. Heckmann. How many argument digits are needed to produce n result digits? In *RealComp '98 Workshop (June 1998 in Indianapolis)*, volume 24 of *Electronic Notes in Theoretical Computer Science*, 2000. URL: `http://www.elsevier.nl/locate/entcs/volume24.html`.
31. C. M. Hoffmann. The problems of accuracy and robustness in geometric computation. *IEEE Comput.*, 22(3):31–41, 1989.
32. C. Y. Hu, T. Maekawa, E. C. Shebrooke, and N. M. Patrikalakis. Robust interval algorithm for curve intersections. *Computer-Aided Design*, 28(6/7):495–506, 1996.

33. C. Y. Hu, N. M. Patrikalakis, and X. Ye. Robust interval solid modeling, part i: representations. *CAD*, 28:807–818, 1996.
34. C. Y. Hu, N. M. Patrikalakis, and X. Ye. Robust interval solid modeling, part ii: boundary evaluation. *CAD*, 28:819–830, 1996.
35. V. Stoltenberg-Hansen J. Blanck and J. V. Tucker. Domain representations of partial functions, with applications to spatial objects and constructive volume geometry. *Theoretical Computer Science*. To appear.
36. D. Jackson. Boundary representation modeling with local tolerances. In *ACM Symposium on Solid Modeling and Applications*, pages 247–253, 1995.
37. M. Karasick, D. Lieber, and L. R. Nackman. Efficient Delaunay triangulation using rational arithmetic. *ACM Trans. Graphics*, 10:71–91, 1991.
38. A. A. Khanban, A. Edalat, and A. Lieutier. Delaunay triangulation and Voronoi diagram with imprecise input data. Submitted. Available from `http://www.doc.ic.ac.uk/~khanban/`.
39. M. Konecny. *Many-valued Real Functions Computable by Finite Transducers using IFS Representations*. PhD thesis, School of Computer Science, University of Birmingham, 2000. Available via URL `http://www.cs.bham.ac.uk/~axj/former-students.html`.
40. V. Menissier-Morain. Arbitrary precision real arithmetic: design and algorithms. submitted to *J. Symbolic Computation*, 1996.
41. T. Ottmann, G. Thiemt, and C. Ullrich. Numerical stability of geometric algorithms. In *Proceeding of the 3rd ACM Annual Symposium on Computational Geometry*, pages 119–125, 1987.
42. D. Plume. A calculator for exact real number computation. Available from `http://www.dcs.ed.ac.uk/home/mhe/plume/index.html`, 1998.
43. P. J. Potts. Efficient on-line computation of real functions using exact floating point. Available from: `http://www.purplefinder.com/~potts`, 1997.
44. P. J. Potts. *Exact Real Arithmetic Using Mobius Transformations*. PhD thesis, Imperial College, 1998. Available from: `http://www.purplefinder.com/~potts`.
45. P. J. Potts and A. Edalat. Exact Real Arithmetic based on Linear Fractional Transformations, December 1996. Draft, Imperial College, available from `http://www-tfm.doc.ic.ac.uk/~pjp`.
46. P. J. Potts and A. Edalat. Exact Real Computer Arithmetic, March 1997. Department of Computing Technical Report DOC 97/9, Imperial College, available from `http://theory.doc.ic.ac.uk/~ae`.
47. P. J. Potts, A. Edalat, and M. Escardó. Semantics of exact real arithmetic. In *Twelfth Annual IEEE Symposium on Logic in Computer Science*. IEEE, 1997.
48. M. B. Pour-El and J. I. Richards. *Computability in Analysis and Physics*. Springer-Verlag, 1988.
49. T. W. Sederberg and R. T. Farouki. Approximation by interval Bezier curves. *IEEE Comput. Graph. Appl.*, 15(2):87–95, 1992.
50. M. Segal. Using tolerances to guarantee valid polyhedral modeling results. *Computer Graphics*, 24(4):105–114, 1990.
51. K. Sugihara. A simple method for avoiding numerical errors and degeneracy in Voronoi diagram construction. *IEICE Trans. Fundamentals*, 1992.
52. K. Sugihara. A robust and consistent algorithm for intersecting convex polyhedra. In *Computer Graphics Forum, EUROGRAPHICS'94*, pages C–45–C–54, 1994.
53. K. Sugihara. Experimental study on acceleration of an exact-arithmetic geometric algorithm. In *Proceeding of the International Conference on Shape Modeling and Applications*, pages 160–168, 1997.

54. K. Sugihara. How to make geometric algorithms robust. *IEICE Trans. Inf. & Syst.*, E833-D(3):447–454, 2000.
55. K. Sugihara and M. Iri. Construction of the Voronoi diagram for one million generators in single-precision arithmetic. *Proc. IEEE*, 80:1471–1484, 1992.
56. K. Sugihara and M. Iri. A robust topology-oriented incremental algorithm for Voronoi diagrams. *International Journal of Computational Geometry and Applications*, pages 179–228, 1994.
57. J. E. Vuillemin. Exact real computer arithmetic with continued fractions. *IEEE Transactions on Computers*, 39(8):1087–1105, 1990.
58. K. Weihrauch. *Computability*, volume 9 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.
59. K. Weihrauch. A foundation for computable analysis. In D.S. Bridges, C.S. Calude, J. Gibbons, S. Reeves, and I.H. Witten, editors, *Combinatorics, Complexity, and Logic*, Discrete Mathematics and Theoretical Computer Science, pages 66–89, Singapore, 1997. Springer-Verlag. Proceedings of DMTCS'96.
60. C. Yap. A geometric consistency theorem for a symbolic perturbation theorem. In *Proc. Fourth ACM Symp. on Computer Geometry*, pages 134–142, June 1988.
61. C. K. Yap. The exact computation paradigm. In D. Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*. World Scientific, 1995.