Final report on EPSRC grant GR/L43077/01

Exact Real Number Computation Using Linear Fractional Transformations

http://www.doc.ic.ac.uk/exact-computation/

Principal Investigator: Abbas Edalat*

1 Summary

The overall aim of the project, as stated in the original Research Proposal, was successfully achieved; the research results actually far surpass the initial objectives. We developed a framework for exact real arithmetic based on linear fractional transformations (LFT), which unifies the signed binary, the continued fractions and the domain-theoretic approaches to exact arithmetic. We designed and implemented algorithms for all elementary functions in this framework in Haskell and in a C library, available on the WEB, and analysed their space and time complexity. The resulting package is distinguished as a well-documented feasible framework for exact arithmetic with a space and time complexity analysis which is computational adequate, in other words provably correct: the user specifies the accuracy with which a particular numerical computation involving the composition of any number of basic elementary function (i.e. arithmetic, algebraic and transcendental functions) is required and the package is guaranteed to return a decimal number which meets the specified degree of accuracy. In order to produce a far more efficient package which can be effectively used in several fundamental applications such as root-finding, numerical integration and solution of differential equations, we then combined our library of continued fraction algorithms for elementary functions with Müller's iRRAM for exact arithmetic, which is indeed the fastest existing package on selective input data, but for which there is no proof of computational adequacy.

We went far beyond the stated original objectives of the project by two significant breakthroughs in areas not envisaged in the project proposal: (i) We developed a domain-theoretic framework for computational geometry/solid modelling, which gives the first proper data type for geometric computation and has so far resulted in robust algorithms with exact real input or imprecise input for the convex hull and the Voronoi diagram/Delaunay triangulation. (ii) We also obtained a domain-theoretic model for differential calculus, which has provided the first proper data type for solving ordinary differential equations up to any degree of accuracy.

2 The Unifying LFT Framework

In order to produce a feasible and computationally adequate framework for exact arithmetic, we synthesized the previous work in computer and real arithmetic in the following three main areas : (i) In Wiedmer's approach [38], that of Boehm and Cartwright [1], implemented by Valerie Menissier [27] and by Dave Plume [31] computable real numbers are approximated by redundant B-adic numbers. (ii) Gosper [15], Vuillemin [36] and Nielsen/Kornerup [29] developed a representation of real numbers by continued fractions, which was implemented by Lester [25]. (iii) In early and mid 90's, research on domain theory and semantics of real number computation by Di Gianantonio [3] and Escardó [14] led to extensions of PCF with a real number data type.

*Department of Computing, Imperial College, London SW7 2BZ. Email: ae@doc.ic.ac.uk

We first constructed an efficient representation of real numbers using composition of LFT's acting on the one-point compactification of the real line with base interval $[0, \infty]$ [10, 34]. An infinite sequence of LFT compositions give a shrinking nested sequence of rational intervals converging to the real number it represent. There is a close relation between such compositions and continued fractions, since any continued fraction expansion

$$r = a_0 + \cfrac{b_0}{a_1 + \cfrac{b_1}{a_2 + \cfrac{b_2}{a_3 + \ddots}}}$$

of a positive real number, with $a_i, b_i \geq 0$, can be expressed as the composition of LFT's $f_0 f_1 f_2 \cdots$, with $f_i(x) = a_i + \dfrac{b_i}{x}$ for $0 \leq i \leq n$. Using the theory of Padé approximation for continued fractions, Potts in his PhD thesis [33] obtained algorithms for all elementary functions in this so-called LFT framework for exact arithmetic.

We then developed a similar computationally adequate framework which takes the well-known and standard signed binary system as the representation of real numbers. By switching the base interval from $[0, \infty]$ to the standard $[-1, 1]$, we combined the signed binary system and the LFT framework for computing all elementary functions [5]. Thus we synthesized the simplicity of the signed binary representation with the elegance and computational power of the LFT framework in one unified system, which we now describe in more detail.

In the LFT framework for exact arithmetic with base interval $[-1, 1]$, a real number r can be presented as the intersection $\{r\} = \bigcap_{n \geq 0} f_0 f_1 f_2 \cdots f_n[-1, 1]$, for a sequence of bounded LFT's with integer coefficients $f_i : [-1, 1] \to \mathbb{R} : x \mapsto \frac{a_i x + c_i}{b_i x + d_i}$, represented by the matrix

$$\begin{pmatrix} a_i & c_i \\ b_i & d_i \end{pmatrix}$$

such that f_i is arbitrary but each f_i, with $i \geq 1$, refines the base interval $[-1, 1]$. The latter condition ensures that the sequence of intervals $f_0 f_1 f_2 \cdots f_n[-1, 1]$ $(n \geq 0)$ is nested, thus giving increasingly better approximations to r. Representing LFT's by matrices, we get a finite or infinite product of matrices. Here are two infinite products for π and e which are obtained from their continued fractions:

$$\pi = \begin{pmatrix} -2 & 2 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -1 & -3 \\ 0 & 4 \end{pmatrix} \begin{pmatrix} -5 & -13 \\ 3 & 15 \end{pmatrix} \cdots \qquad e = \begin{pmatrix} 1 & 3 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 6 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 10 \end{pmatrix} \cdots$$

A rational number can be presented by a finite product in which the last matrix is singular, i.e. a vector or a 0-dimensional LFT, representing a rational number in $[-1, 1]$.

Function are expressed using 2-dimensional LFT's. A 2-dimensional bounded LFT is a mapping $f : [-1, 1] \times [-1, 1] \times \mathbb{R} \to \mathbb{R}$ where

$$f(x, y) = \frac{axy + cx + ey + g}{bxy + dx + fy + h} \quad \text{has integer coefficients represented by the } \textit{tensor}\text{:} \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix}$$

We say f is refining if $f([-1, 1], [-1, 1]) \subseteq [-1, 1]$. By choosing the coefficients of a tensor appropriately, one can perform addition, multiplication and division. For example, addition corresponds to choosing $a = b = d = f = g = 0$ and $c = e = h = 1$; Figure 1(left). The sum of $\pi + e$ is given by the *expression tree* in Figure 1(right). After composing the tensor (2-LFT) with the two matrices (1-LFT) on the first level, we obtain the first approximation $[17/3, 27/3]$ to $\pi + e$. Since all remaining matrices are refining, we get a shrinking sequence of nested intervals, the next one being $[223/39, 565/96]$, giving better and better approximations to $\pi + e$. Any basic elementary function, such as $\sin x$, $\tan x$ and $\exp x$ can be expressed by as a binary expression tree whose nodes are 2-dimensional LFT's (tensors) and the leaves are x as in Figure 2(left). For example the expression tree in Figure 2(right) computes $\tan(1/\pi)$, using the above matrix product of π and giving the first two approximations as $[2/9, 2/5]$ and $[294/907, 35/101]$.

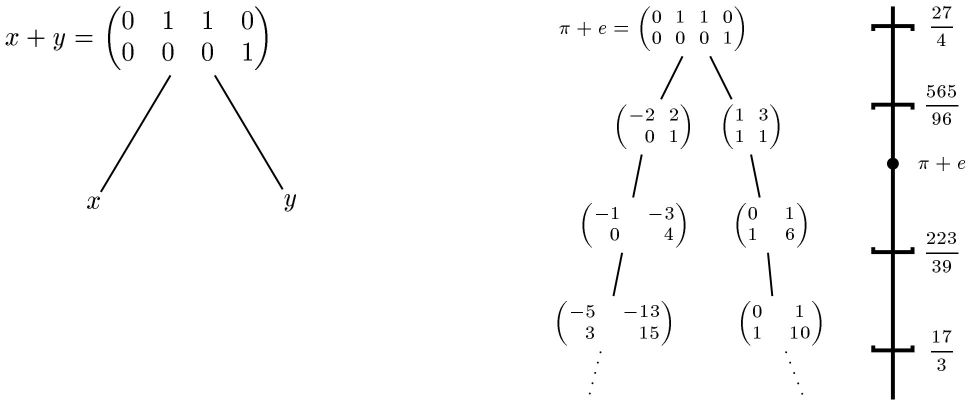

Figure 1: Expression tree for addition (left) and $\pi + e$ (right)

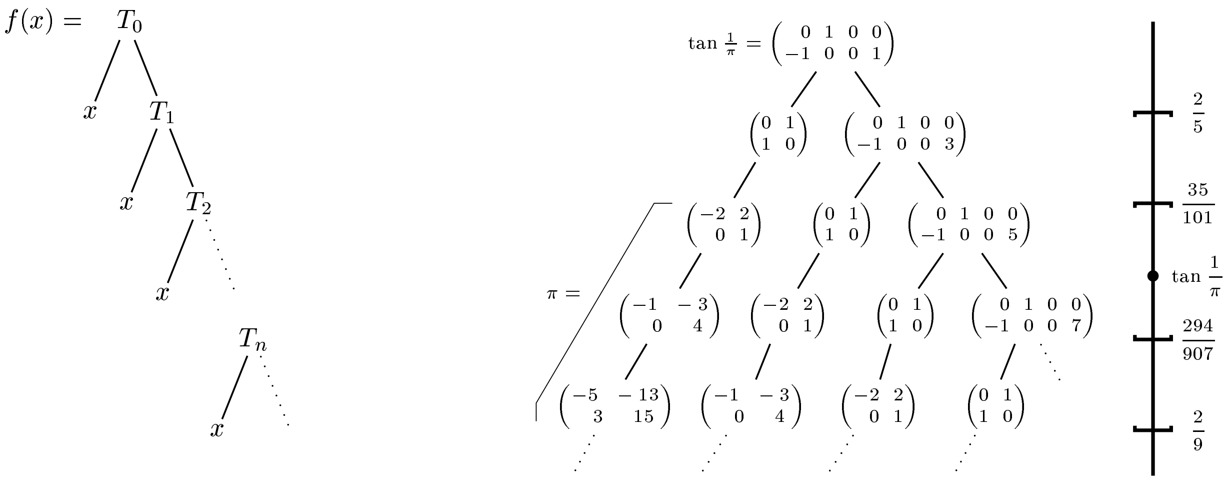

Figure 2: Expression tree for transcendental functions (left) and $\tan \frac{1}{\pi}$ (right)

In practice, however, we would like the real number representing the input and output of a computation be of a standard form which would allow only a controlled growth in the size of integers representing the rational intervals and a uniform contraction of intervals suitable for space and time complexity analysis. It is thus natural to use the *signed binary system* with digits $d \in \{-1, 0, 1\}$, where the digit -1 is also written as $\overline{1}$. The redundancy in the representation is needed for exact arithmetic. Numbers in $[-1, 1]$ are then represented by digit streams $\xi = d_1 d_2 \cdots$ representing $[\xi] = \sum_{i=1}^{\infty} d_i 2^{-i}$. Finite sequences correspond to dyadic intervals: $\delta = d_1 d_2 \cdots d_n$ representing $[\delta] = [\sum_{i=1}^{n} d_i 2^{-i} - 2^{-n}, \sum_{i=1}^{n} d_i 2^{-1} + 2^{-n}]$, with the empty string $(n = 0)$ denoting the interval $[-1, 1]$. We have $[d : \xi] = A_d([\xi])$ where A_d denotes the affine function with $A_d(x) = \frac{x+d}{2}$, corresponding to the *digit matrices*

$$\begin{pmatrix} 1 & d \\ 0 & 2 \end{pmatrix} \qquad \text{with } d \in \{-1, 0, 1\}.$$

Multiplication by a power of 2 will give us the representation of all numbers. Thus, we arrive at the representation $(e \parallel \xi)$ where e is an integer (called *exponent*) and ξ is a digit stream (called *mantissa*), and $(e \parallel \xi)$ represents $[(e \parallel \xi)] = 2^e \cdot [\xi]$. Semantically, the attachment of the exponent can again be captured by an affine map, namely $[(e \parallel \xi)] = E^e([\xi])$, where E^e is given by $E^e(x) = 2^e \cdot x$, corresponding to the *exponent matrices*

$$\begin{pmatrix} 2^e & 0 \\ 0 & 1 \end{pmatrix}.$$

This number system is so similar to the floating point numbers that we call it "exact floating point". Since A_d and E_e are affine, they are actually LFT's and thus the signed binary expansion is in particular a finite or infinite product of LFT's. We therefore require that all input and output of any computation be a signed binary expansion.

An expression tree is then evaluated into a signed binary expansion up to any desired accuracy as follows: (i) Obtain an upper bound for the exponent. (ii) Refine the exponent until it is sufficiently small to give the required precision or until the representation is normalised, i.e. the mantissa starts with $\overline{1}\overline{1}$, $\overline{1}0$, 10 or 11, which implies that it represents a number in $[-1, -1/4] \cup [1/4, 1]$. (iii) Compute the digits of the mantissa up to the required precision. This three step computation is done by *absorbing* and *emitting* exponent or digit matrices into or from matrices or tensors. Whereas absorption of digit matrices, which corresponds to extracting information from the argument of a function, is always allowed, emission of digit matrices from a tensor or a matrix, which corresponds to extracting information from the function itself, is only permitted if the tensor or the matrix is refining; this ensures that the sequence of output intervals are nested. The global strategy for a lazy evaluation of an expression tree is the following: emit a matrix if you can, otherwise absorb from left or right using a fair strategy. The lazy evaluation scheme is incremental in the sense that once a result is obtained with a given accuracy, a more accurate result can be provided by starting with the current state of the expression tree. Lazy evaluation is, however, not efficient. In practice, we use an $\epsilon - \delta$ analysis: a good lower (rather than upper) bound estimate for the number of digits that need to be absorbed in mass in order to emit one or more digits. When digits totalling this lower bound have been absorbed, either an emission is already possible or another lower bound estimate is calculated. Since, in practice, only one or two lower bound estimates will enable an emission, this scheme, extended to mass emission, gives an efficient way of obtaining the output.

Several variations of the LFT framework were shown to be *computationally adequate* when presented in extensions of PCF with a real number data type; the first such extension was given in [34]. Computational adequacy means that the sequence of shrinking, nested rational intervals computed at the finite stages of a real program actually converges to the real number the program represents. Previous extensions of PCF with real numbers [3, 4, 14], use the *parallel conditional* (PIF) construct which is based on comparison of real numbers with zero, an undecidable predicate. In recursive calls, this construct produces intractable algorithms. In search of an alternative, the *redundant if* or RIF was put forward, which compares a real number with two distinct rational numbers, ensuring that one of the comparisons succeeds in finite time; several extensions of PCF with RIF were shown to be adequate [32, 11]. We have also shown [13] that the domain-theoretic approach to computability on the real line leads to the same notion of computability as in the classic work of Pour-El and Richards [35] and Weihrauch [37].

3 Space and Time Complexity

Heckmann analysed the space and time complexity of the LFT approach. It was established that in matrices and tensors, the bit-size of the entries generally grows linearly with the number of absorbed and emitted digits, even if all common factors are cancelled out [17]. In studying the time complexity of real number operations, the *contractivities* of the LFT's play an important role [19]. Contractivities allow us to estimate the number of input digits which are needed to compute a certain number of output digits [21]. With this information, one can obtain lower and upper bounds for the number of LFT operations needed to perform calculations. The results about the growth of the entries of the LFT's then allow us to estimate the complexity of the individual LFT operations.

A few special functions can be computed by algorithms that produce n digits of the result in time $O(n)$, assuming that the digits of the argument are already computed and freely available. These are functions which can be realised by LFT's with a constant upper bound for their integer entries. An interesting example of this kind is the absolute value function $x \mapsto |x|$ which can be computed with an intentional algorithm not based on an expression tree.

Calculating n digits of the value of a function represented by integer matrices with linear bit-size growth, has complexity $O(n^2)$ if digits are handled one by one. The reason is that after n output digits have been produced, at least one entry of the representing matrix has bit size $O(n)$ so that the calculation of the next digit has complexity $O(n)$. This quadratic complexity can be reduced to the complexity, $O(n \log n \log\log n)$, of multiplication of two n-bit integers, by replacing the digit by digit approach with strategies to deal with many digits at once. An algorithm for the consumption of many argument digits in one go was already implemented by Potts. It can be complemented by an algorithm for the production of many output digits at once [18].

Later, Heckmann found a systematic way to derive LFT expansions from Taylor series, which are accessible for a larger range of functions [20]. The LFT's derived from continued fractions are usually more contractive than those coming from Taylor series, so that less LFT operations are needed to obtain some output accuracy. On the other hand, the Taylor LFT's are simpler than the LFT's coming from continued fractions, and their entries grow more slowly. This means that the individual LFT operations can be performed more quickly. Run-time experiments seem to indicate that these two effects approximately cancel out each other.

4 Root-finding, Integration, Solution of Differential Equations

We also developed an efficient algorithm [12] for root-finding in exact arithmetic using a trisection of intervals that can be compared with the well-known bisection method in numerical computation, which actually fails in exact arithmetic since comparison of real numbers is undecidable.

In search of feasible methods for integration, we showed that the classical techniques in numerical integration (namely the compound trapezoidal and Simpson's rules and the Gauss-Legendre formulae) can be implemented in an exact real arithmetic framework in which the numerical value of an integral of an elementary function is obtained up to any desired accuracy without any round-off errors. Any exact framework which provides a library of algorithms for computing elementary functions with an arbitrary accuracy is suitable for such an implementation; we used the exact real arithmetic framework based on LFT's to implement these integration algorithms in Haskell. We also showed that the Euler and Runge-Kutta methods for solving the initial value problem of an ordinary differential equation can be implemented using an exact framework which will guarantee the convergence of the approximation to the actual solution of the differential equation as the step size in the partition of the interval in question tends to zero [6, 16].

5 Implementation

Functional languages, including Caml, Miranda and Haskell were used extensively to develop prototype implementations of the LFT framework with base interval $[0, \infty]$. These prototypes were written by Potts, Errington and Krznarić. All necessary data types, including data types for linear fractional transformations (LFT's), expression trees and exact floating point, have been implemented, as well as all arithmetic operations and transcendental functions. These implementations are fully lazy and are based on simple types, accessible to users who have limited knowledge about the LFT approach. They differ largely in their level of abstraction. Reals are infinite objects and must necessarily be evaluated lazily. Some of the implementations made use of the laziness available in the underlying language, while others implemented abstract machines in which the operational semantics was made explicit.

We then wrote a library in C, which provides an efficient package and has broader appeal since it would not require users to learn a new language. The most recent version, version 5.0, available at the Exact Computation web-page at Imperial College is approximately 15,000 lines of C, and was written by Lindsay Errington. It consists of a base layer which implements the LFT framework, a collection of primitive arithmetic functions and analytic functions on both real and rational arguments. The integer entries in the LFT's can become arbitrarily large. For this reason, the library uses the GNU Multiple Precision Arithmetic Library (GMP) to manipulate large integers. Briefly, using the primitives in the library, a programmer constructs a representation of a real-valued expression as a graph in the heap. Each LFT is an object in the heap which points to its arguments. The graph denotes a real number. The library is continuation based. The heap structure and the abstract machine can be seen as specialisations of those used for implementing general purpose functional programming languages.

Although the LFT framework for exact arithmetic is computationally adequate and reasonably efficient, Müller's iRRAM for exact arithmetic [28] (likewise Lester's new package), based on floating point intervals and approximation of elementary functions by Taylor series, is far more efficient. It computes the function on a floating point intervals containing the real number input and repeats the procedure with a finer interval until the output interval is within the required degree of accuracy. Although there is no proof that this framework is computationally adequate, i.e. that it does indeed converge on every input, it is the fastest known exact arithmetic package on test input data. We combined Müller's framework with our library of continued fraction algorithms for elementary functions and obtained an even more efficient package in C++ for applications such as root-finding, integration and numerical solution of differential equations where the same function has to be computed repeatedly at a great number of input points.

Furthermore, Gratrix [16] developed a prototype implementation of the framework in Mathematica (cf. Kelsey's Maple package [22]), whereas Christodoulides has developed a calculator with a Java interface based on the C-library. The calculator can compute all elementary functions and perform root-finding and numerical integration for these functions. All the above implementations are accessible from our web-page: `http://www.doc.ic.ac.uk/exact-computation/`.

6 Models for Geometry and Differential Calculus

The early achievement of the aims of the project and an active collaboration with André Lieutier, a senior engineer of Dassault Systemes, a French CAD company allowed us to go further and make two significant breakthroughs in areas not initially envisaged in the research proposal.

(i) We developed a domain-theoretic framework for computational geometry, solid modelling and CAD systems which gives a methodology for constructing robust geometric algorithms both with exact real number input and with imprecise input data [8]. In this framework we obtained an $O(N \log N)$ algorithm for the convex hull in 2d and 3d, which is the first robust algorithm that can handle exact real number input or imprecise input data and has the same complexity as the best classical algorithms for the convex hull, that are known to be non-robust for such input.

Given N rational rectangles as approximations to N points in the plane, the algorithm finds the inner and outer convex hull of these rectangles as in Figure 3(left).

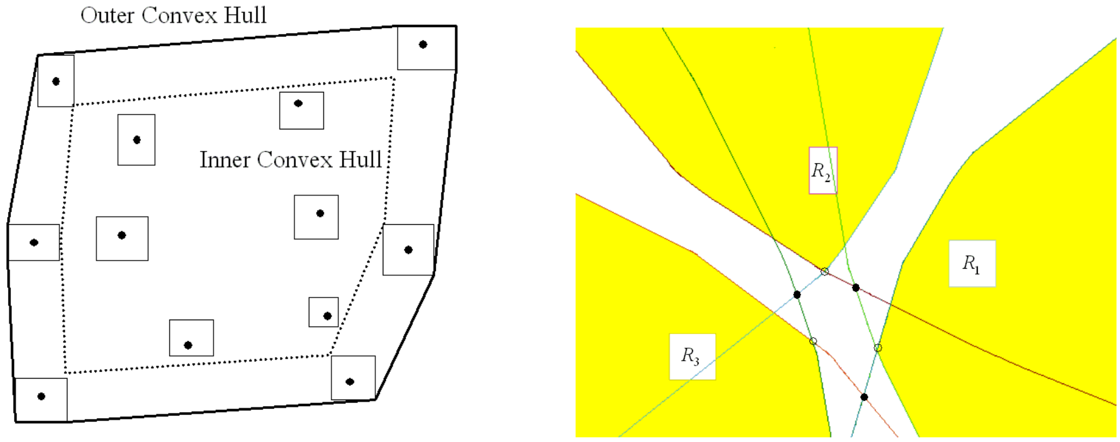

Figure 3: Inner and outer convex hull of rectangles (left); Voronoi diagram of 3 rectangles (right)

Any point in the inner convex hull is guaranteed to be inside the convex hull of any set of N points approximated by these rectangles, whereas any point in the outer convex hull is guaranteed to be outside the convex hull of any set of N points approximated by these rectangles. In the limit as the rectangles shrink to points the inner and outer convex hulls tend to the convex hull of the points and, for computable points, the convergence is effective both in the Hausdorff metric and in the Lebesgue measure [9].

Similarly, in this framework we obtained a random incremental algorithm, with average complexity $O(N \log N)$ for Delaunay triangulation in 2d and, consequently, an algorithm for Voronoi diagram in 2d with average complexity $O(N \log N)$ [23]. Again these are the first robust algorithms for Delaunay triangulation and Voronoi diagram which can cope with exact real number input or imprecise input data and have the same complexity as the best classical algorithms that are in fact non-robust for such input. Figure 3(right) shows the Voronoi diagrams for three imprecise points (rectangles).

(ii) We succeeded to define domains for differentiable functions (C^k and C^∞) which, for the first time, provide us with proper data-types for differential calculus and scientific computing [7]. In particular, we obtained a domain-theoretic extension of Picard's fundamental theorem for the existence and uniqueness of the solution of an initial value problem in the theory of ordinary differential equations. This extension theorem on domains allows us to solve for the solution of a differential equation by alternately updating the information about the function and its derivative. The approximating solutions give, at each stage of computation, piecewise rational polynomials as upper and lower bounds to the exact solution. Figure 4 shows this for $\frac{dx}{dt} = 2t + x$ with $x(.5) = 4.5$. The dashed rectangles, including the inner dashed rectangle for the initial value, give the first approximations to the function (top) and its derivative (bottom): the graphs of these functions will be contained in the respective rectangles in the corresponding horizontal intervals. The regions with solid boundaries give the new approximations as a result of updating the information about first the derivative (bottom) and then the function (top). On the left the result is shown after one round of updating; on the right, the result, for a smaller dashed rectangle, is shown after two rounds of updating. The lower and upper approximations tend in the limit to the exact solution $x = 7.5\ e^{t-\frac{1}{2}} - 2t - 2$. We have a proto-type implementation of this in Haskell.

7 Training, Dissemination and Further Work

Dr Peter Potts wrote his PhD thesis [33] on the subject of the project; Marko Krznarić and Ali Khanban are taking their transfer from MPhil to PhD based on their work, respectively, on the

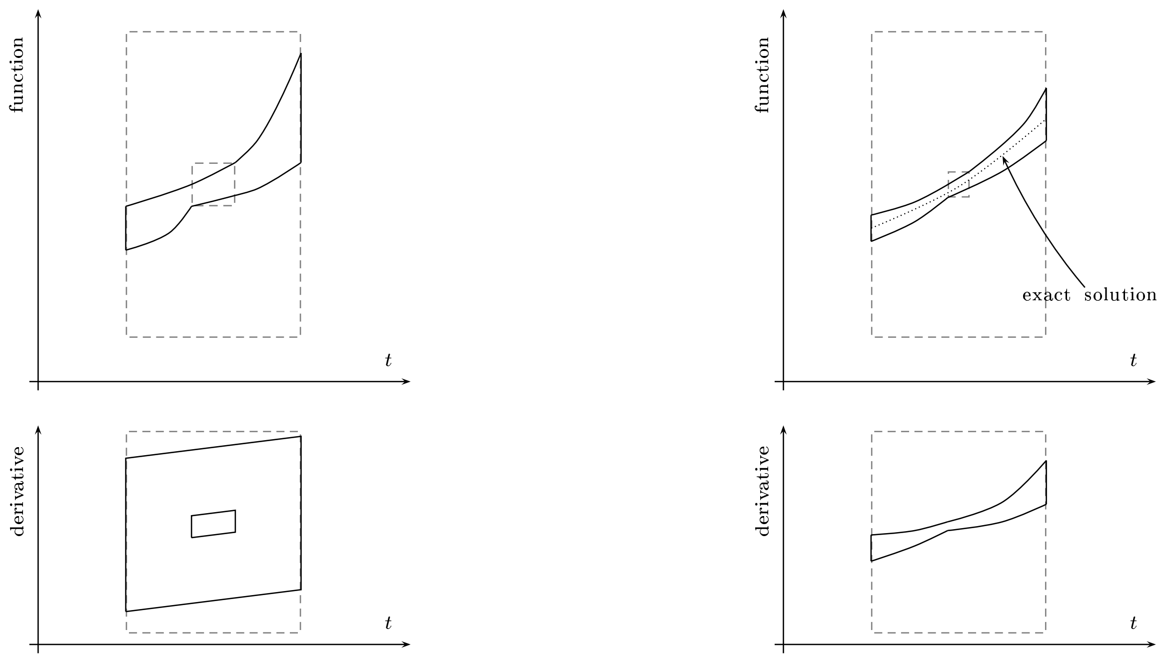

Figure 4: Two approximations to the solution of $\frac{dx}{dt} = 2t + x$ with $x(.5) = 4.5$

domain-theoretic model for solving differential equations and on domain-theoretic model of Voronoi diagrams and Delaunay triangulation. There have been five MSc dissertations [24, 16, 2, 26, 30] on the work of the project and three undergraduate projects. Lindsay Errington, Martin Escardó and Philipp Sünderhauf, who were employed on other grants at Imperial College, actually made significant contributions to the research on this project and, in the case of Errington, to the C implementation of exact arithmetic.

The work of the project was presented as *invited talks* and *key-note speeches* at the following conferences and meetings: (i) the 2nd International Symposium on Domain Theory, Chengdu, China, October 2001 (three hour lecture course), (ii) Workshop on Pisot Numbers and Real Number Computations, Rome, October 2001 (iii) the International Conference on Applicable General Topology, Ankara, Turkey, August 2001, (iv) the First Irish Conference on the Mathematical Foundations of Computer Science and Information Technology, Cork, July 2000, (v) ETAPS 2000, Berlin, March 2000, (vi) ICALP'99, Prague, July 1999, (vii) the 14th Summer Conference on General Topology and its Applications, Long Islands, August 1999, (viii) the British Colloquium for Theoretical Computer Science, March 1998, St-Andrews.

The work of the project was also presented at numerous other conferences and workshops, as in the list of the published papers, as well as in several Dagstuhl and Appsem workshops and at the Mittag-Lefler Institute's Symposium on Logic and Types in June 2001. To date, four journal papers on the project have appeared or accepted for publication. The subject of the project was a main theme at APPSEM Summer School in Portugal September, 2000. The 70 page invited article [5] is an expanded version of the lecture notes at the summer school and gives a comprehensive account of the findings of the project. A complete list of the papers and implementations of the project can be accessed from the Exact Computation web-page at `http://www.doc.ic.ac.uk/exact-computation/`.

We are now collaborating with Müller to further combine our two packages for exact arithmetic. We are also developing the work on the domain-theoretic model for computational geometry and differential calculus. The project got industrial support (£60,000 over three years) from Dassault Systemes (Aix-en-Province) to develop proper data types for geometric modelling in CAD.

References

[1] 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.

[2] N. Boudris. Computing roots of polynomials in exact real arithmetic. Master's thesis, Department of Computing, Imperial College, 2000.

[3] P. Di Gianantonio. *A functional approach to real number computation.* PhD thesis, University of Pisa, 1993.

[4] P. Di Gianantonio. Real number computability and domain theory. *Information and Computation*, 127(1):11–25, May 1996.

[5] A. Edalat and R. Heckmann. Computing with real numbers: (i) LFT approach to real computation, (ii) Domain-theoretic model of computational geometry. In Gilles Barthe, Peter Dybjer, Luis Pinto, and Joao Saraiva, editors, *LNCS*. Springer, 2001. To appear.

[6] A. Edalat and M. Krznarić. Numerical integration with exact real arithmetic. In Jiří Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming*, volume 1644 of *LNCS*, pages 90–104. Springer, 1999. 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 1999.

[7] A. Edalat and A. Lieutier. Domain theory and differential calculus. Submitted. Available from `http://www.doc.ic.ac.uk/~ae/`.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] A. Edalat and F. Rico. Root finding in exact real arithmetic using linear fractional transformations. In *Third Real Numbers and Computers Conference*, pages 27–44, Paris, 1998.

[13] A. Edalat and P. Sünderhauf. A domain theoretic approach to computability on the real line. *Theoretical Computer Science*, 210:73–98, 1998.

[14] M. H. Escardó. PCF extended with real numbers. *Theoretical Computer Science*, 162(1):79–115, August 1996.

[15] W. Gosper. *Continued Fraction Arithmetic.* HAKMEM Item 101B, MIT Artificial Intelligence Memo 239. MIT, 1972.

[16] S. Gratrix. Exact real arithmetic in numerical computation. Master's thesis, Department of Computing, Imperial College, 1999.

[17] 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.

[18] 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`.

[19] 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.

[20] R. Heckmann. Translation of Taylor series into LFT expansions. Submitted to Proceedings of Dagstuhl Seminar "Symbolic Algebraic Methods and Verification Methods", November 1999.

[21] 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`.

[22] T. Kelsey. Maple-pvs. Available from `http://www-theory.dcs.st-and.ac.uk/~tom/`, 2001.

[23] 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/`.

[24] M. Krznarić. Numerical integration in exact real arithmetic. Master's thesis, Department of Computing, Imperial College, 1998.

[25] D. R. Lester. Vuillemin's exact real arithmetic. In R. Heldal, C. K. Holst, and P. L. Wadler, editors, *Functional Programming, Glasgow 1991: Proceedings of the 1991 Workshop, Portree, UK*, pages 225–238, Berlin, 1992. Springer-Verlag.

[26] P. Manousopoulos. Boolean polygon operations and rounding in the domain-theoretic framework for solid modelling. Master's thesis, Department of Computing, Imperial College, 2000.

[27] V. Menissier-Morain. Arbitrary precision real arithmetic: design and algorithms. submitted to *J. Symbolic Computation*, 1996.

[28] N. Müller. The irram: Exact arithmetic in c++. In *Proceedings of the fourth workshop of Computability and Complexity in Analysis, Swansea 2000*, 2000. Full paper to appear in LNCS.

[29] A. Nielsen and P. Kornerup. MSB-first digit serial arithmetic. *J. of Univ. Comp. Scien.*, 1(7):523–543, 1995.

[30] A. Nikolopoulos. Boolean and isometric operations on rational polyhedra. Master's thesis, Department of Computing, Imperial College, 2001.

[31] D. Plume. A calculator for exact real number computation. Available from `http://www.dcs.ed.ac.uk/home/mhe/plume/index.html`, 1998.

[32] P. J. Potts. Efficient on-line computation of real functions using exact floating point. Available from: `http://www.purplefinder.com/~potts`, 1997.

[33] P. J. Potts. *Exact Real Arithmetic Using Mobius Transformations*. PhD thesis, Imperial College, 1998. Available from: `http://www.purplefinder.com/~potts`.

[34] 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.

[35] M. B. Pour-El and J. I. Richards. *Computability in Analysis and Physics*. Springer-Verlag, 1988.

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

[37] K. Weihrauch. A simple introduction to computable analysis. Technical Report 171-2/1995, FernUniversität, 1995.

[38] E. Wiedmer. Computing with infinite objects. *Theoretical Computer Science*, 10:133–155, 1980.