# A Golden Ratio Notation for the Real Numbers

Pietro Di Gianantonio

*dip. di Matematica e Informatica, Università di Udine*
*via delle Scienze 206 I-33100 Udine Italy*
*email: pietro@dimi.uniud.it*

**Abstract**

Several methods to perform exact computations on real numbers have been proposed in the literature. In some of these methods real numbers are represented by infinite (lazy) strings of digits. It is a well known fact that, when this approach is taken, the standard digit notation cannot be used. New forms of digit notations are necessary. The usual solution to this representation problem consists in adding new digits in the notation, quite often negative digits. In this article we present an alternative solution. It consists in using non natural numbers as "base", that is, in using a positional digit notation where the ratio between the weight of two consecutive digits is not necessarily a natural number, as in the standard case, but it can be a rational or even an irrational number. We discuss in full detail one particular example of this form of notation: namely the one having two digits (0 and 1) and the golden ratio as base. This choice is motivated by the pleasing properties enjoyed by the golden ratio notation. In particular, the algorithms for the arithmetic operations are quite simple when this notation is used.

*AMS Subject Classification (1991):* 65Y99.

*CR Subject Classification (1991):* F.2.1, G.1.0.

*Keywords & Phrases:* real number computability, digit notation, golden ratio.

*Note:* Work funded by the Human Capital and Mobility fellowship program EuroFocs.

## 1. Introduction

In the ordinary practice, computations on real numbers are performed by approximating real numbers by a subset of the rational numbers and by approximating the arithmetic on real numbers by a limited precision arithmetic on rationals. This is not the only possible way to compute on real numbers. Also a form of exact computation is feasible. In this case the results of the computations can be obtained with arbitrary precision. The exact computation on real numbers has been studied by several authors. Here we just mention Boehm and Cartwright ([2, 3]), Ko ([9]), Martin-Löf ([12]), Vuillemin ([16]) and Weihrauch ([17]).

In [2] and [6] an approach to real number computation via lazy functional languages is considered. In this approach a real number is represented by an infinite lazy sequence of digits. A real function is implemented by a lazy program that receives as input a lazy stream of digits, the argument, and generates as output a lazy stream representing the result. The computation proceeds as follows: the program examines a finite number of digits of the argument, starting from the most meaningful ones. When the argument is known with a sufficient precision some digits of the output are generated, then some more digits of the input are examined in order to compute new digits of the output, and so on. Each digit of the output needs to be correct. The exact result is the limit of the computation.

Lazy functional languages are not the only possible approach to computability on infinite streams. Several authors used the digit notation for reals and a different approach to the computability, as for example Turing machine in [15], [17] [18] or approximations spaces in [14].

The standard digit notation is not suitable for the lazy computation described above. The usual solution to this problem consists in adding new digits to the notation, often negative digits. In this work we present an alternative solution. It consists in employing a different form of notation for real numbers, namely a digit notation where the "base" is not a natural number. That is we use a positional digit notation where the ratio between the weight of two consecutive digits is not necessarily a natural number, as in the standard case, but it can be a real number. Among all possible notations of this form we choose one that is particularly interesting. It uses two digits and the golden ratio as base. The reason for this choice is that this notation permits simple algorithms for the arithmetic operations.

The outline of the paper is as follows. In section 2 we present several forms of digit notation and we explain why the standard notation cannot be used for exact computation on reals. In section 3 the golden base notation is introduced. In section 4 we present the algorithms for the arithmetic operations working on the golden base notation. In section 5 we investigate the problem of whether there exist other bases for a binary notation which permit simple algorithms for the arithmetic operations.

## 2. Digit Notations for Reals.

In analysis the real numbers are defined in a plethora of different ways: Cauchy sequences of rational numbers, Cauchy sequences of decimal rationals, Dedekind cuts in the field of rationals, infinite decimal expansions, and so on. Classically all these notations are equivalent and we can study analysis without worrying about which notation for real numbers we are currently using. If we intend to compute on real numbers this is not any more true and some care has to be taken in choosing the notation: for instance Dedekind cuts and Cauchy sequences turn out not to be equivalent notations. In this paper we concentrate on the digit notations for real numbers. Also in this case some care has to be taken in the notation used. The classical digit notation of real numbers can be defined as follows:

**Definition 1** *Given a natural number $b > 1$ a digit notation with base $b$ of a real number $x$ is given by a sequence of integers $z_0 : z_1 : z_2 : \ldots$, such that:*
*i)* $\forall i \geq 1 \,.\, 0 \leq z_i < b$
*ii)* $x = \sum_{i \in \mathbb{N}} z_i \times b^{-i}$

Here the integer part of a real is denoted by a single integer number instead of using a finite string of digits. We make this change to the classical notation for simplicity reasons. None of the results we present is affected by this change.

As mentioned in the introduction a simple method to compute on real numbers consists in using a lazy functional language. In this approach a real number is implemented by a program that lazily generate a stream of integers denoting it. A real number is said *computable* if it can be implemented by a program. It is not difficult to prove that this notion of computability is independent from the base number. A function between real numbers is implemented by a program that receives as input lazy streams (or, more precisely, programs generating lazy streams) denoting the arguments and generates as output a lazy stream denoting the result. A function is said *computable*, in a given notation, if it can be implemented by a program. A problem here arises. All notations in Definition 1, independently from the base used, are not suitable for real number computation. Using them even the most fundamental functions such as addition or multiplication are not computable.

Here is a simple example that illustrates the inadequacy of the standard decimal notation. We show that *no algorithm can compute multiplication by* 3. A hypothetical algorithm for this function will not be able to generate the first digit of the result when it receives as input the stream $0 : 3 : 3 : 3 \ldots$ denoting the number $1/3$. The result of the multiplication is 1. 1 can be denoted in two ways, namely by $1 : 0 : 0 : 0 \ldots$ or by $0 : 9 : 9 : 9 \ldots$ . If the algorithm generates 1 as the first digit, this must happen after the algorithm has examined a finite number of digits of the argument. Let us suppose that the first

$n$ digits have been examined before generating 1. Then the algorithm generates 1 as first digit also when it receives as input the stream $0:\overbrace{3:3:\ldots:3}^{n}:0:0\ldots$. But this in incorrect since the exact result should then be a stream having the first digit equal to 0. An analogous argument can be made if the algorithm generates 0 as first digit.

Similar examples show that also the other arithmetic operations are not computable. Clearly the problem presented above is not caused by the choice of 10 as base number. The same problem would arise for any other base.

A more general explanation for the inadequacy of the standard digit notation is the following: in a digit notation all the streams starting with 0 denote numbers contained in the closed interval $[0,1]$ and all the streams starting with 1 denote numbers contained in the closed interval $[1,2]$. To generate the first digit of a real number $x$ we have therefore to decide whether $x$ is a number smaller or bigger than 1. Consider the case of $x$ being the result of an arithmetic function with argument $y$. If we know a sufficient number of digits of $y$ we are able to approximate $x$ with an arbitrary precision (that is, given an arbitrary natural number $n$ we are able to determine a rational number $a$ s. t. $a-1/n < x < a+1/n$), but still it can be the case that we are not able to decide whether $x$ is smaller or bigger than 1 and so we are not able to generate the first digit of $x$.

The introduction of *signed-digits*, is a simple way to overcome these difficulties. The standard interpretation can be extended to streams of positive and negative digits. For example the string $0:+4:-5:-3:+2$ denotes the rational number $(+4\times10^{-1})+(-5\times10^{-2})+(-3\times10^{-3})+(+2\times10^{-4})$. Formally we have the following definition

**Definition 2** *Given a natural number $b>1$ a signed-digit notation with base $b$ of a real number $x$ is given by a sequence of integers $z_0:z_1:z_2:\ldots$, such that:*
*i)* $\forall i \geq 1\,.\ -b < z_i < b$
*ii)* $x = \sum_{i\in\mathbb{N}} z_i \times b^{-i}$

The negative digit notation was first proposed by Leslie [11] and independently by Cauchy [5]. This notation has also been proposed for hardware implementation in order to avoid the propagation of the carry [1].

Going back to the previous example we show how the introduction of signed-digits solves the computational difficulties. The algorithm for multiplication by 3 can safely generate 1, as first digit, after having read the first two digits of the stream $0:3:\ldots$. Observe that if the input becomes $0:3:-9:-9\ldots = 1/5$ it is still possible to generate the exact result by the stream $1:-4:0:0:0:\ldots = 3/5$. If the input becomes $0:3:9:9:9:\ldots = 2/5$ the output can become $1:2:0:0:0:\ldots$ .

With the signed-digits notation a stream starting with 0 denotes a real number contained in the interval $[-1,1]$ while a stream starting with 1 denotes a real number contained in the interval $[0,2]$. In this notation in order to generate the first digit of a real number $x$ it is not necessary to decide whether $x$ is smaller or bigger than 1. To generate the first digit of $x$ it is sufficient to know its value with precision $1/2$. If $x$ is the result of an arithmetic function then to approximate its value with precision $1/2$ is sufficient to examine just a finite number of digits of the function's arguments. Perfectly similar considerations hold for the successive digits, to generate each of them it is sufficient to know the value of $x$ with a given precision and this can always be done examining a finite number of digits of the arguments. It follows that with the signed-digit notation all the arithmetic functions are computable.

It is possible to prove that all the signed-digit notations (independently from the base used) are computationally equivalent, in the sense that they characterize the same class of computable reals and computable real functions. In fact for any pair of signed-digit notations there exists an effective

translation. That is, there exists a program that, when it receives as input a stream denoting a real $x$ in the first notation, gives as output a stream denoting the same real $x$ in the second notation. Therefore we can define a real function *computable* if it is computable w.r.t. one signed-digit notation.

It is interesting to notice that also if the the standard notations and the signed-digit notations are different with respect to the computability of functions, nevertheless they characterize the same class of computable reals.

## 3. Golden base notation.

The simplest signed digit notation is the one having base 2 and using three digits $\{-1, 0, 1\}$. In this section we consider the problem of whether there exist notions for the real numbers making use of just two digits, 0 and 1, and suitable for the real number computation. It turns out that such notations exist, moreover there are countably many of them. We single out, among these, one that we think to be optimal w.r.t. the simplicity of the algorithms for the arithmetic operations.

A preliminary remark here is appropriate. By the words *base of a digit notation* we mean the ratio between the weights of two consecutive digits in the notation. More explicitly: if the base is $b$, the finite string of digits $a_0 : \ldots : a_n$ denotes the number $\sum_{i \leq n} a_i / b^i$. Normally base and number of digits in a notation coincide. The distinction between the two concepts is therefore often blurred. But nothing prevents us from defining notations where base and number of digits are different, like in the signed-digit notation. And nothing prevents us from choosing as base a number that is not natural. *Rational* or even *irrational* numbers can legitimately be chosen as bases. The only restriction is that the base has to be a (computable) number strictly larger than 1 and smaller than the number of digits in the notation.

Exploiting this idea it is possible to define a collection of real number notations.

**Definition 3** *Given natural number $d$ and a computable real number $b$ with $1 < b < d$, a digit representation with $d$ digits and base $b$ of a real number $x$ is given by a sequence of integers $z_0 : z_1 : z_2 : \ldots$, such that:*
*i)* $\forall i \geq 1 \,.\, 0 \leq z_i < d$
*ii)* $x = \sum_{i \in \mathbb{N}} z_i \times b^{-i}$

A notation for reals of the above form will be called *real-base notation*. The use of irrational numbers as base is not new. For example, real-base notations have been already studied in [13]. The real-base notation can be generalized further. If the base of a digit notation is an imaginary number then a string of digits represents a complex number. Every complex number can then be represented by a single string of digits, if we choose a suitable base. Moreover using an imaginary base notation it is possible to obtain algorithms for the arithmetic operations on complex numbers that are quite similar to the standard algorithms on real numbers. An historical excursion on different number notations can be found in Knuth [8].

It is possible to prove that all real-base notations are computationally equivalent to the signed-digit notations. Again the prove can be given exhibiting, for any possible pair of notations, a program implementing an effective translation between them.

It follows that there are infinitely many notations using two digits and suitable for real number computation. Brouwer was probably the first one to notice the computational difficulties of the standard binary notation. He suggested as solution the use of a binary notation with base 3/2 [4]. In corollary 13 we prove that the binary notation with base 3/2 is not the most convenient choice in order to obtain simple algorithms for the arithmetic operations. We claim that the simplest algorithms for the arithmetic operations are obtained when the value of the base is the golden ratio, that is the number $\phi = (\sqrt{5} + 1)/2$. An intuitive and partial explanation for this fact is the following. With the

golden ratio as base, the stream $0:1:1:0:0:0:\ldots$ denotes the number 1 (here with $0:0:0\ldots$ we indicate the stream with all elements equal to 0). It follows that in the golden ratio notation it is possible to rewrite a group of digits in a stream without altering the value denoted by the stream. For example the number 1 is denoted by each one of the following streams: "$1:0:0:0\ldots$", "$0:1:1:0:0:0\ldots$", "$0:1:0:1:1:0:0:0\ldots$". Exploiting identities of this kind it is possible to obtain algorithms for the arithmetic operations. The golden ratio is the only admissible value for a base, for which the identity $1:0:0:0\ldots = 0:1:1:0:0:0\ldots$ holds.

It is interesting to observe that in the fractional point notation, with the golden ratio as base, each natural number can be represented by a finite string of digits. In fact, the natural number 1 is represented by the string 1.0 but it is also represented by the string 0.11. It follows that the natural number 2 is represented by 1.11 ($1.11 = 1.00 + 0.11 = 1 + 1$). Again we can rewrite 1.11 as 10.01 and therefore the natural number 3 is represented by the string 11.01 or by the string 100.01. Going on in this way we have $4 = 101.01 = 100.1111$, $5 = 101.1111 = 110.0111$, $6 = 111.0111 = 1010.0001$, $7 = 1011.0001 = 1100.0001$ and so on.

In the following we use a representation for the reals slightly different from the one given in Definition 3. In that representation the first digit is an arbitrary integer (with no limitation) and represent the integer part of the real number denoted by the stream. It is a kind of fixed-point representation. The representation we are going to define is more on the spirit of a floating-point (exponent-mantissa) representation. It has the advantage that the algorithms for the arithmetic operations are simpler to define with it. We will use the following notation: the greek letters $\alpha$, $\beta$ and $\gamma$ will range on binary streams (streams of binary digits), $\alpha_i$ will denote the $i+1$ digit of $\alpha$ and $\alpha_{|_i}$ will denote the substream $\alpha_i:\alpha_{i+1}:\ldots$.

**Definition 4** *In the golden notation the stream $z:\alpha$, with $z$ integer number and $\alpha$ binary stream ($\forall i \in I\!N\,.\,\alpha_i \in \{0,1\}$), denotes the real number $[\![z:\alpha]\!]_f$ defined by:*

$$[\![z:\alpha]\!]_f = (-1 + \sum_{i \in I\!N} \alpha_i \times \phi^{-i-1}) \times \phi^{2z}$$

The first term $-1$ in the above formula has been introduced in order to represent negative numbers. Note that the more conventional notation with sign, mantissa and exponent cannot be used. In fact in order to establish the sign of a number we have to decide if the number is smaller or bigger that 0. Using the arguments presented at page 2 one can prove that no algorithm for addition can always decide the sign of its result.

## 4. Algorithms for Arithmetic Operations.

In the following we describe the algorithms for the arithmetic operations in the golden ratio notation. For each operation we define two algorithms. The first one works on a simplified notation for the real numbers. The second one works on the full notation of Definition 4. The simplified notation uses only binary streams to represent reals.

**Definition 5** *In the simplified notation the binary stream $\alpha = \alpha_0:\alpha_1:\ldots$ denotes the real number $[\![\alpha]\!]_s$ defined by:*

$$[\![\alpha]\!]_s = \sum_{i \in I\!N} \alpha_i \times \phi^{-i-1}$$

Using the simplified notation we can denote just a subset of the real numbers: all real numbers contained in the interval $[0, \phi]$. This interval is not closed for the arithmetic operations therefore an

overflow problem arises. To solve this problem the algorithms for the simplified notation yield results of the arithmetic operations divided by fixed factor, $\phi$ or $\phi^2$ depending from the operations.

The algorithms are described by a set of rewriting rules over streams of digits. The notation used is quite similar to the one used in the lazy functional programming language Miranda [7] to define functions on lazy streams. The algorithms for the full notation derive from the ones for the simplified notation.

*4.1 Addition*

The algorithm executes the computation from left to right (from the most meaningful digits to the least meaningful ones) and uses a carry consisting of two digits. At each step, the algorithm needs to examine at most two digits of each addend in order to determine which step to execute.

Formally we define a function $A$ having the following behaviour:
i) The arguments of $A$ are two binary streams $\alpha$ and $\beta$ (the addends), and two binary digits $a, b$ (the carry).
ii) The following equality holds:

$$[\![A(\alpha,\, \beta,\, a,\, b)]\!]_s = ([\![\alpha]\!]_s + [\![\beta]\!]_s + a/\phi + b/\phi^2)/\phi^2$$

**Definition 6** *The function $A$ implementing the addition in the simplified notation is defined by the following set of rewriting rules:*

$$\begin{array}{lcl}
A(0{:}\alpha,\, 0{:}\beta,\, 0,\, b) & \Rightarrow & 0{:}A(\alpha,\, \beta,\, b,\, 0) \\
A(0{:}0{:}\alpha,\, 0{:}\beta,\, 1,\, b) & \Rightarrow & 0{:}A(b{:}\alpha,\, \beta,\, 1,\, 1) \\
A(0{:}1{:}\alpha,\, 0{:}1{:}\beta,\, 1,\, 1) & \Rightarrow & 1{:}0{:}A(\alpha,\, \beta,\, 0,\, 1) \\
A(0{:}0{:}\alpha,\, 1{:}0{:}\beta,\, 1,\, 0) & \Rightarrow & 0{:}1{:}A(\alpha,\, \beta,\, 1,\, 0) \\
A(0{:}\alpha,\, 1{:}\beta,\, 1,\, 1) & \Rightarrow & 1{:}A(\alpha,\, \beta,\, 0,\, 0) \\
A(1{:}\alpha,\, 1{:}\beta,\, 1,\, b) & \Rightarrow & 1{:}A(\alpha,\, \beta,\, b,\, 1) \\
 & & \\
A(1{:}\alpha,\, 0{:}\beta,\, a,\, b) & \Rightarrow & A(0{:}\alpha,\, 1{:}\beta,\, a,\, b) \\
A(a{:}\alpha,\, 1{:}\beta,\, 0,\, b) & \Rightarrow & A(a{:}\alpha,\, 0{:}\beta,\, 1,\, b) \\
A(a_1{:}1{:}\alpha,\, a_2{:}0{:}\beta,\, a_3,\, b) & \Rightarrow & A(a_1{:}0{:}\alpha,\, a_2{:}1{:}\beta,\, a_3,\, b) \\
A(a_1{:}b{:}\alpha,\, a_2{:}1{:}\beta,\, a_3,\, 0) & \Rightarrow & A(a_1{:}b{:}\alpha,\, a_2{:}0{:}\beta,\, a_3,\, 1)
\end{array}$$

The interesting part of the algorithm is described by the first 6 rewriting rules, the last 4 rules just permute digits having equal weights. It is interesting to observe that in the above set of rewriting rules there is a duality between 0 and 1. If we substitute in all rules each digit 0 with 1 and vice-versa we obtain an equivalent set of rules for addition.

**Proposition 1** *i) $A$ is a well defined function: for every $\alpha, \beta, a, b$ there is one and only one stream $\gamma$ such that: $A(\alpha,\, \beta,\, a,\, b) \Rightarrow \gamma$.*

*ii) $A$ has the intended behaviour, that is:*

$$[\![A(\alpha,\, \beta,\, a,\, b)]\!]_s = ([\![\alpha]\!]_s + [\![\beta]\!]_s + a/\phi + b/\phi^2)/\phi^2$$

**Proof.** i) It is not difficult to prove by induction and case analysis that for each natural number $n$ and for each $\alpha, \beta, a, b$ there exists a unique string $c_0 : \ldots : c_{n-1}$ having length $n$ and there exists $\alpha', \beta', a', b'$ such that: $A(\alpha,\, \beta,\, a,\, b) \Rightarrow c_0{:}\ldots{:}c_{n-1}{:}A(\alpha',\, \beta',\, a',\, b')$.

ii) We prove by induction on the natural number $n$ that for any pair of binary streams $\alpha, \beta$ and for any pair of binary digits $a, b$ we have:

$$| \, [\![A(\alpha,\, \beta,\, a,\, b)]\!]_s - ([\![\alpha]\!]_s + [\![\beta]\!]_s + a/\phi + b/\phi^2)/\phi^2 \, | \leq \, \phi^{1-n}$$

The base step of the induction can be proved by a simple calculation.

Next we need to prove that if the inequality holds for $n$ then it holds also for $n+1$. The proof of this inductive step is done by case analysis. One proves that the inductive step is valid for each possible value of the first two digits of the addends $\alpha$, $\beta$ and for each possible value of the two digits of carry $a$, $b$. Here we consider the case when $\alpha = 0{:}\alpha_{|_1}$, $\beta = 1{:}\beta_{|_1}$, $a = 1$ and $b = 1$. In this case the fifth rewriting rule has to be applied. We can write:
$| \, [\![A(0{:}\alpha_{|_1},\, 1{:}\beta_{|_1},\, 1,\, 1)]\!]_s - ([\![0{:}\alpha_{|_1}]\!]_s + [\![1{:}\beta_{|_1}]\!]_s + 1/\phi + 1/\phi^2)/\phi^2 \, |$
$= | \, [\![1{:}A(\alpha_{|_1},\, \beta_{|_1},\, 0,\, 0)]\!]_s - ([\![\alpha_{|_1}]\!]_s/\phi + 1/\phi + [\![\beta_{|_1}]\!]_s/\phi + 1)/\phi^2 \, |$
$= | \, 1/\phi + [\![A(\alpha_{|_1},\, \beta_{|_1},\, 0,\, 0)]\!]_s/\phi - (\phi + [\![\alpha_{|_1}]\!]_s/\phi + [\![\beta_{|_1}]\!]_s/\phi)/\phi^2 \, |$
$= | \, [\![A(\alpha_{|_1},\, \beta_{|_1},\, 0,\, 0)]\!]_s - ([\![\alpha_{|_1}]\!]_s + [\![\beta_{|_1}]\!]_s)/\phi^2 \, | \, /\phi$
$\leq \phi^{1-n}/\phi = \phi^{1-(n+1)}$ by inductive hypothesis.

The proofs for the other cases can be reduced to this one or follow a similar pattern. □

We present the rewriting rules of the function $A$ in a different form. In doing so we show how our algorithm for the addition can be executed with pen and paper and we highlight the similarities between our algorithm and the standard one we learned at primary school. Differently form the standard algorithm, our algorithm proceeds from left to right (from the most meaningful digits to the less meaningful ones) and uses a carry composed by two digits. It is described by the rules:

$$\begin{array}{r|l} & 0\ a \\ + & 0\,\alpha_1\,\alpha_2\ldots \\ = & 0\,\beta_1\,\beta_2\ldots \\ \hline & \ldots c'c\ --\end{array} \quad \Rightarrow \quad \begin{array}{r|l} & a\ \ 0 \\ + & \alpha_1\,\alpha_2\,\alpha_3\ldots \\ = & \beta_1\,\beta_2\,\beta_3\ldots \\ \hline & \ldots c'c\,0\,--\end{array}$$

$$\begin{array}{r|l} & 1\ 0 \\ + & 0\,\alpha_1\,\alpha_2\ldots \\ = & 0\,\beta_1\,\beta_2\ldots \\ \hline & \ldots c'c\ --\end{array} \quad \Rightarrow \quad \begin{array}{r|l} & 1\ \ 1 \\ + & \alpha_1\,\alpha_2\,\alpha_3\ldots \\ = & \beta_1\,\beta_2\,\beta_3\ldots \\ \hline & \ldots c'c\,0\,--\end{array}$$

$$\begin{array}{r|l} & 1\,1 \\ + & 0\,1\,\alpha_2\ldots \\ = & 0\,1\,\beta_2\ldots \\ \hline & \ldots c'c\ --\end{array} \quad \Rightarrow \quad \begin{array}{r|l} & 0\ \ 1 \\ + & \alpha_2\,\alpha_3\,\alpha_4\ldots \\ = & \beta_2\,\beta_3\,\beta_4\ldots \\ \hline & \ldots c'c\,1\,0\,--\end{array}$$

$$\begin{array}{r|l} & 1\,0 \\ + & 1\,0\,\alpha_2\ldots \\ = & 0\,0\,\beta_2\ldots \\ \hline & \ldots c'c\ --\end{array} \quad \Rightarrow \quad \begin{array}{r|l} & 1\ \ 0 \\ + & \alpha_2\,\alpha_3\,\alpha_4\ldots \\ = & \beta_2\,\beta_3\,\beta_4\ldots \\ \hline & \ldots c'c\,0\,1\,--\end{array}$$

$$\begin{array}{r|l} & 1\ \ 1 \\ + & 1\,\alpha_1\,\alpha_2\ldots \\ = & 0\,\beta_1\,\beta_2\ldots \\ \hline & \ldots c'c\ --\end{array} \quad \Rightarrow \quad \begin{array}{r|l} & 0\ \ 0 \\ + & \alpha_1\,\alpha_2\,\alpha_3\ldots \\ = & \beta_1\,\beta_2\,\beta_3\ldots \\ \hline & \ldots c'c\,1\,--\end{array}$$

$$\begin{array}{r|l} & 1\ a \\ + & 1\,\alpha_1\,\alpha_2\ldots \\ = & 1\,\beta_1\,\beta_2\ldots \\ \hline & \ldots c'c\ --\end{array} \quad \Rightarrow \quad \begin{array}{r|l} & a\ \ 1 \\ + & \alpha_1\,\alpha_2\,\alpha_3\ldots \\ = & \beta_1\,\beta_2\,\beta_3\ldots \\ \hline & \ldots c'c\,1\,--\end{array}$$

In these rules the top two digits are the carry, the next two streams are the parts of the addends not yet examined, the string under the line is the result generated so far. As usual, digits on the same column have equal weight. The two dashes "−−" stand for two digits of the result not yet generated. This means that the last generated digit of the result has a weight that is a factor $\phi^3$ bigger than the one of the firsts digits of the addends not yet examined. For the sake of brevity we did not write the rules that permute digits having equal weight. From another point of view we can think that each one of the given rules subsumes all the rules that can be obtained from it by permuting digits having identical weitght.

We sketch here an informal proof of correctness for the algorithm. The proof consists in showing that each rule is correct. For example we can justify the fifth rule by:

$$
\begin{array}{l|l|l}
 & \begin{array}{r} 1\ 1 \\ +\ \ 1\,\alpha_1\,\alpha_2\ldots \\ =\ \ 0\,\beta_1\,\beta_2\ldots \end{array} & \\
\hline
 & \ldots c'c\ --\ \ \ \ & \\
\hline
 & \begin{array}{r} 1\ 0\ 0 \\ +\ \ 1\,\alpha_1\,\alpha_2\ldots \\ =\ \ 0\,\beta_2\,\beta_2\ldots \end{array} & \\
\Rightarrow & \hline \ldots c'c\ --\ \ \ \ & \text{since } 001 = 100 \\
\hline
 & \begin{array}{r} 1\ 1\ 0 \\ +\ \ 0\,\alpha_1\,\alpha_2\ldots \\ =\ \ 0\,\beta_2\,\beta_2\ldots \end{array} & \\
\Rightarrow & \hline \ldots c'c\ --\ \ \ \ & \text{by associativity and commutativity of addition} \\
\hline
 & \begin{array}{r} 1\ 0\ 0\ 0 \\ +\ \ 0\,\alpha_1\,\alpha_2\ldots \\ =\ \ 0\,\beta_2\,\beta_2\ldots \end{array} & \\
\Rightarrow & \hline \ldots c'c\ --\ \ \ \ & \text{since } 001 = 100 \\
\hline
 & \begin{array}{r} 0\ \ 0 \\ +\ \ \alpha_1\,\alpha_2\,\alpha_3\ldots \\ =\ \ \beta_1\,\beta_2\,\beta_3\ldots \end{array} & \\
\Rightarrow & \hline \ldots c'c\,1\ --\ \ & \\
\end{array}
$$

All other rules can be justified in a similar way.

**Definition 7** *The function $A'$ implementing addition in the full notation is defined by:*

$$
\begin{array}{lcl}
A'(z{:}\alpha,\ z{:}\beta) & \Rightarrow & (z{+}1){:}A(\alpha,\ \beta,\ 1,\ 0) \\
A'(z{:}\alpha,\ t{:}\beta) & \Rightarrow & A'((z{+}1){:}1{:}0{:}\alpha,\ t{:}\beta) \ \textit{if}\ z < t \\
A'(z{:}\alpha,\ t{:}\beta) & \Rightarrow & A'(z{:}\alpha,\ (t{+}1){:}1{:}0{:}\beta) \ \textit{if}\ t < z
\end{array}
$$

The correctness of the above algorithm follows easily from the lemma:

**Lemma 2** *For every integer $z$ and binary streams $\alpha$, $\beta$ the following two equalities hold:*

$$[\![A'(z{:}\alpha,\ z{:}\beta)]\!]_f = [\![z{:}\alpha]\!]_f + [\![z{:}\beta]\!]_f$$

$$[\![z{:}\alpha]\!]_f = [\![(z{+}1){:}1{:}0{:}\alpha]\!]_f$$

**Proof.**

$$\begin{aligned}
[\![A'(z{:}\alpha,\, z{:}\beta)]\!]_f &= [\![(z{+}1){:}A(\alpha,\, \beta,\, 1,\, 0)]\!]_f \\
&= (-1 + [\![A(\alpha,\, \beta,\, 1,\, 0)]\!]_s) \times \phi^{2z+2} \\
&= (-1 + ([\![\alpha]\!]_s + [\![\beta]\!]_s + 1/\phi)/\phi^2) \times \phi^{2z+2} \\
&= (-\phi^2 + [\![\alpha]\!]_s + [\![\beta]\!]_s + 1/\phi) \times \phi^{2z} \\
&= (-\phi - 1 + [\![\alpha]\!]_s + [\![\beta]\!]_s + 1/\phi) \times \phi^{2z} \\
&= (-1 - 1/\phi - 1 + [\![\alpha]\!]_s + [\![\beta]\!]_s + 1/\phi) \times \phi^{2z} \\
&= (-1 + [\![\alpha]\!]_s) \times \phi^{2z} + (-1 + [\![\beta]\!]_s) \times \phi^{2z} \\
&= [\![z{:}\alpha]\!]_f + [\![z{:}\beta]\!]_f
\end{aligned}$$

$$\begin{aligned}
[\![(z{+}1){:}1{:}0{:}\alpha]\!]_f &= (-1 + [\![1{:}0{:}\alpha]\!]_s) \times \phi^{2z+2} \\
&= (-1 + 1/\phi + [\![\alpha]\!]_s/\phi^2) \times \phi^{2z+2} \\
&= (-\phi^2 + \phi + [\![\alpha]\!]_s) \times \phi^{2z} \\
&= (-1 + [\![\alpha]\!]_s) \times \phi^{2z} \\
&= [\![z{:}\alpha]\!]_f
\end{aligned}$$

□

*4.2 Subtraction.*
We first define a function $C$ that "complements" a binary stream:

**Definition 8** *The function $C$ is defined by:*

$$\begin{array}{lcl}
C(1{:}\alpha) & \Rightarrow & 0{:}C(\alpha) \\
C(0{:}\alpha) & \Rightarrow & 1{:}C(\alpha)
\end{array}$$

**Proposition 3** *For every binary stream $\alpha$*

$$[\![C(\alpha)]\!]_s = \phi - [\![\alpha]\!]_s$$

**Proof.** It follows from the equalities:

$$[\![\alpha]\!]_s + [\![C(\alpha)]\!]_s = \sum_{i \in I\!N^+} 1/\phi^i = 1/(1 - \phi^{-1}) = \phi$$

□

**Definition 9** *The function $C'$ that evaluates the inverse of a real number is defined by:*

$$\begin{array}{lcl}
C'(z{:}\alpha) & \Rightarrow & (z{+}1){:}C_1(\alpha) \\
C_1(0{:}\alpha) & \Rightarrow & 1{:}1{:}0{:}C(\alpha) \\
C_1(1{:}0{:}\alpha) & \Rightarrow & 1{:}0{:}C_1(\alpha) \\
C_1(1{:}1{:}\alpha) & \Rightarrow & 1{:}0{:}0{:}1{:}C(\alpha)
\end{array}$$

**Proposition 4** *For every integer number $z$ and binary stream $\alpha$ we have;*

$$[\![C'(z{:}\alpha)]\!]_f = -[\![z{:}\alpha]\!]_f$$

**Proof.** It is easy to check that the proposition is true if for every binary stream $\alpha$:

$$-1 + [\![C_1(\alpha)]\!]_s = -(-1 + [\![\alpha]\!]_s)/\phi^2$$

that is:

$$[\![C_1(\alpha)]\!]_s = 1 + 1/\phi^2 - [\![\alpha]\!]_s/\phi^2$$

We prove by induction on $n$ that for every natural number $n$ and every stream $\alpha$ we have:

$$\mid [\![C_1(\alpha)]\!]_s - (1 + 1/\phi^2 - [\![\alpha]\!]_s/\phi^2) \mid \leq \phi^{1-n}$$

The base step can be proved by a simple calculation. The inductive step can be proved by case analysis on the first two digits of $\alpha$. Here we consider the case $\alpha = 1{:}0{:}\alpha_{|_2}$, we have:
$\mid [\![C_1(1{:}0{:}\alpha_{|_2})]\!]_s - (1 + 1/\phi^2 - [\![1{:}0{:}\alpha_{|_2}]\!]_s/\phi^2) \mid$
$= \mid [\![1{:}0{:}C_1(\alpha_{|_2})]\!]_s - 1 - 1/\phi^2 + 1/\phi^3 + [\![\alpha_{|_2}]\!]_s/\phi^4 \mid$
$= \mid 1/\phi + [\![C_1(\alpha_{|_2})]\!]_s/\phi^2 - (1/\phi + 1/\phi^2) - (1/\phi^3 + 1/\phi^4) + 1/\phi^3 + [\![\alpha_{|_2}]\!]_s/\phi^4 \mid$
$= \mid [\![C_1(\alpha_{|_2})]\!]_s - (1 + 1/\phi^2 - [\![\alpha_{|_2}]\!]_s/\phi^2) \mid /\phi^2$
$\leq \phi^{1-n}/\phi^2 < \phi^{1-(n+1)}$ (by inductive hypothesis).

The other cases are simpler. □

We give here also an algorithm for subtraction that does not use the inverse function $C'$.

**Definition 10** *The function $S'$ implementing the subtraction in the full notation is defined by:*

$$\begin{array}{lcl} S'(z{:}\alpha,\, z{:}\beta) & \Rightarrow & (z{+}1){:}A(\alpha,\, C(\beta),\, 1,\, 1) \\ S'(z{:}\alpha,\, t{:}\beta) & \Rightarrow & S'((z{+}1){:}1{:}0{:}\alpha,\, t{:}\beta) \text{ if } z < t \\ S'(z{:}\alpha,\, t{:}\beta) & \Rightarrow & S'(z{:}\alpha,\, (t{+}1){:}1{:}0{:}\beta) \text{ if } t < z \end{array}$$

The correctness of the algorithm derives from the following lemma:

**Lemma 5** *For every integer $z$ and binary streams $\alpha$, $\beta$ the following equality holds:*

$$[\![S'(z{:}\alpha,\, z{:}\beta)]\!]_f = [\![z{:}\alpha]\!]_f - [\![z{:}\beta]\!]_f$$

**Proof.**

$$\begin{array}{rl} [\![S'(z{:}\alpha,\, z{:}\beta)]\!]_f & = [\![(z{+}1){:}A(\alpha,\, C(\beta),\, 1,\, 1)]\!]_f \\ & = (-1 + [\![A(\alpha,\, C(\beta),\, 1,\, 1)]\!]_s) \times \phi^{2z+2} \\ & = (-1 + ([\![\alpha]\!]_s + [\![C(\beta)]\!]_s + 1)/\phi^2) \times \phi^{2z+2} \\ & = (-1 + ([\![\alpha]\!]_s + \phi - [\![\beta]\!]_s + 1)/\phi^2) \times \phi^{2z+2} \\ & = (-1 + ([\![\alpha]\!]_s - [\![\beta]\!]_s + \phi^2)/\phi^2) \times \phi^{2z+2} \\ & = ([\![\alpha]\!]_s - [\![\beta]\!]_s) \times \phi^{2z} \\ & = (-1 + [\![\alpha]\!]_s) \times \phi^{2z} - (-1 + [\![\beta]\!]_s) \times \phi^{2z} \\ & = [\![z{:}\alpha]\!]_f - [\![z{:}\beta]\!]_f. \end{array}$$

□

*4.3 Multiplication.*
An easy way to obtain an algorithm for multiplication is to reduce multiplication to a series of additions.

**Definition 11** *The function $P$ implementing the multiplication in the simplified notation is defined by:*

$$
\begin{array}{lcl}
P(0{:}\alpha,\ \beta) & \Rightarrow & 0{:}P(\alpha,\ \beta)) \\
P(\alpha,\ 0{:}\beta) & \Rightarrow & 0{:}P(\alpha,\ \beta)) \\
P(1{:}0{:}\alpha,\ 1{:}0{:}\beta) & \Rightarrow & 0{:}A(A(\alpha,\ \beta,\ 0,\ 0),\ 0{:}P(\alpha,\ \beta),\ 1,\ 0) \\
P(1{:}1{:}\alpha,\ 1{:}0{:}\beta) & \Rightarrow & A(A(0{:}\alpha,\ \beta,\ 0,\ 0),\ 0{:}0{:}P(\alpha,\ \beta),\ 1,\ 0) \\
P(1{:}0{:}\alpha,\ 1{:}1{:}\beta) & \Rightarrow & A(A(\alpha,\ 0{:}\beta,\ 0,\ 0),\ 0{:}0{:}P(\alpha,\ \beta),\ 1,\ 0) \\
P(1{:}1{:}\alpha,\ 1{:}1{:}\beta) & \Rightarrow & A(A(\alpha,\ \beta,\ 0,\ 0),\ 0{:}0{:}P(\alpha,\ \beta),\ 1,\ 1)
\end{array}
$$

**Proposition 6** *i) $P$ is a well defined function between binary streams.*
*ii) For every pair of binary streams $\alpha$, $\beta$:*

$$[\![P(\alpha,\ \beta)]\!]_s = \frac{[\![\alpha]\!]_s \times [\![\beta]\!]_s}{\phi^2}$$

**Proof.** The proof is very similar to the corresponding one for addition. We just sketch it. Point i) is easy. To prove point ii) one proves that for that for every natural number $n$ and for every pair of streams $\alpha$, $\beta$ the following equality holds:

$$\mid [\![P(\alpha,\ \beta)]\!]_s - \frac{[\![\alpha]\!]_s \times [\![\beta]\!]_s}{\phi^2} \mid \leq\ \phi^{1-n}$$

This can be proved by induction on $n$ and case analysis on the first two digits of $\alpha$ and $\beta$. □

**Definition 12** *The function $P'$ implementing the product in the full notation is defined by:*

$$P'(z{:}\alpha,\ t{:}\beta) \Rightarrow (z+t+2){:}A(P(\alpha,\ \beta),\ C(A(\alpha,\ \beta,\ 0,\ 0)),\ 1,\ 0)$$

**Proposition 7** *For every pair of integers $z, t$ and for every pair of binary streams $\alpha$, $\beta$ we have:*

$$[\![P'(z{:}\alpha,\ t{:}\beta)]\!]_f = [\![z{:}\alpha]\!]_f \times [\![t{:}\beta]\!]_f$$

**Proof.**
$[\![P'(t{:}\alpha,\ t{:}\beta)]\!]_f$
$= [\![(z+t+2){:}A(P(\alpha,\ \beta),\ C(A(\alpha,\ \beta,\ 0,\ 0)),\ 1,\ 0)]\!]_f$
$= (-1 + [\![A(P(\alpha,\ \beta),\ C(A(\alpha,\ \beta,\ 0,\ 0)),\ 1,\ 0)]\!]_s) \times \phi^{2z+2t+4}$
$= (-1 + ([\![(P(\alpha,\ \beta)]\!]_s + [\![C(A(\alpha,\ \beta,\ 0,\ 0))]\!]_s + 1/\phi)/\phi^2) \times \phi^{2z+2t+4}$
$= (-1 + ([\![\alpha]\!]_s \times [\![\beta]\!]_s)/\phi^2 + (\phi - ([\![\alpha]\!]_s + [\![\beta]\!]_s)/\phi^2) + 1/\phi)/\phi^2) \times \phi^{2z+2t+4}$
$= (-\phi^4 + [\![\alpha]\!]_s \times [\![\beta]\!]_s + \phi^3 - [\![\alpha]\!]_s - [\![\beta]\!]_s + \phi) \times \phi^{2z+2t}$
$= (-\phi^4 + \phi^3 + \phi + [\![\alpha]\!]_s \times [\![\beta]\!]_s - [\![\alpha]\!]_s - [\![\beta]\!]_s) \times \phi^{2z+2t}$
$= ((-\phi^3 - \phi - 1) + \phi^3 + \phi + [\![\alpha]\!]_s \times [\![\beta]\!]_s - [\![\alpha]\!]_s - [\![\beta]\!]_s) \times \phi^{2z+2t}$
$= (-1 + [\![\alpha]\!]_s \times [\![\beta]\!]_s - [\![\alpha]\!]_s - [\![\beta]\!]_s) \times \phi^{2z+2t}$
$= (-1 + [\![\alpha]\!]_s) \times (-1 + [\![\beta]\!]_s) \times \phi^{2z+2t}$
$= [\![z{:}\alpha]\!]_f \times [\![t{:}\beta]\!]_f$. □

The time necessary to evaluate the first $n$ digits of the product using the algorithm described above is proportional to $n^2$. Using the algorithm of Karatsuba ([8]) it is possible to obtained an incremental algorithm for the multiplication having a time complexity of order $n^{1.89}$ ([10]).

*4.4 Division.*
Following the approach used for the other operations we first define an algorithm valid for the simplified notation.

**Definition 13** *The function $D$ implementing the division in the simplified notation is defined by:*

$$\begin{array}{lcl}
D(\alpha,\, 1{:}\beta) & \Rightarrow & D_1(0{:}0{:}\alpha,\, C(\beta)) \\
D_1(0{:}0{:}\alpha,\, \beta) & \Rightarrow & D_2(A(\alpha,\, 0{:}\beta,\, 0,\, 0),\, 0{:}\alpha,\, \beta) \\
D_1(0{:}1{:}\alpha,\, \beta) & \Rightarrow & D_2(A(\alpha,\, 0{:}\beta,\, 1,\, 1),\, 1{:}\alpha,\, \beta) \\
D_1(1{:}0{:}\alpha,\, \beta) & \Rightarrow & D_2(A(\alpha,\, 1{:}\beta,\, 1,\, 1),\, \alpha,\, \beta) \\
D_2(0{:}0{:}\gamma,\, \alpha,\, \beta) & \Rightarrow & 0{:}D_1(\alpha, \beta) \\
D_2(0{:}1{:}0{:}\gamma,\, \alpha,\, \beta) & \Rightarrow & 0{:}D_1(\alpha,\, \beta) \\
D_2(0{:}1{:}1{:}\gamma,\, \alpha,\, \beta) & \Rightarrow & 1{:}D_1(0{:}0{:}\gamma,\, \beta) \\
D_2(1{:}\gamma,\, \alpha,\, \beta) & \Rightarrow & 1{:}D_1(\gamma,\, \beta)
\end{array}$$

As we will prove in the following $[\![D(\alpha,\, \beta)]\!]_s = [\![\alpha]\!]_s/([\![\beta]\!]_s \times \phi)$. There is a restriction on the arguments of the function $D$. $D(\alpha,\, \beta)$ is defined only if $\beta = 1{:}\beta_{|_1}$. This restriction garanties that the result of the division is small enough to be represented in the simplified notation.

The above algorithm is based on the Euclidean algorithm for division. The idea is the following: in order to evaluate the first digit of $[\![\alpha]\!]_s/([\![\beta]\!]_s \times \phi)$ we consider the value $[\![\alpha]\!]_s - [\![\beta]\!]_s$ and we determine if this number is larger than zero or if it smaller than $1/\phi^2$. In the first case we can safely generate 1 as first digit followed by the result of the division of $([\![\alpha]\!]_s - [\![\beta]\!]_s)$ by $[\![\beta]\!]_s$. In the second case we can safely generate 0 as first digit followed by the result of the division of $[\![\alpha]\!]_s$ by $[\![\beta]\!]_s$. The algorithm is defined by few rewriting rules, but its correctness is quite difficult to prove. To reduce the number of rewriting rules employed many arithmetic properties need to be used.

**Proposition 8** *For every pair of binary streams $\alpha$, $\beta$, if $\beta = 1{:}\beta_{|_1}$, we have:*

*i) there exists a unique stream $\gamma$ such that: $D(\alpha,\, \beta) \Rightarrow \gamma$;*

*ii) the following equality holds:*

$$[\![D(\alpha,\, \beta)]\!]_s = \frac{[\![\alpha]\!]_s}{[\![\beta]\!]_s \times \phi}$$

**Proof.** First of all observe there is no rule for the function $D_1$ when the first argument has form $1{:}1{:}\alpha_{|_2}$, so to prove that the algorithm converges we need to prove that such a case never occurs.

**Lemma 9** *For every integer number $n$ and for every pair of binary streams $\alpha$, $\beta$, if $\beta = 1{:}\beta_{|_1}$ then,*
*i) there exist unique $\delta_0{:}\ldots{:}\delta_{n-1}$, $\alpha'$, $\beta'$ s.t.:*
*$D(\alpha, \beta) \Rightarrow \delta_0{:}\ldots{:}\delta_{n-1}{:}D_1(\alpha', \beta')$, moreover,*

- $\alpha' \neq 1{:}1{:}\alpha'_{|_2}$
- $[\![\alpha']\!]_s + [\![\beta']\!]_s/\phi^2 \leq 2/\phi$

*ii) there exist unique $\delta_0{:}\ldots{:}\delta_{n-1}$, $\alpha'$, $\beta'$, $\gamma'$ s.t.,*
*$D(\alpha, \beta) \Rightarrow \delta_0{:}\ldots{:}\delta_{n-1}{:}D_2(\gamma', \alpha', \beta')$ moreover,*

- $\gamma' \neq 1{:}1{:}1{:}\gamma'_{|_3}$
- $[\![\gamma']\!]_s + [\![\beta']\!]_s/\phi^4 \leq 2/\phi$
- *and*
  - *either* $\gamma' = 1{:}\gamma'_{|_1}$

– *either* $\alpha' \neq 1:1:\alpha'_{|_2}$ *and* $[\![\gamma']\!]_s = [\![\alpha']\!]_s/\phi + [\![\beta']\!]_s/\phi^3$

**Proof.** The proof is by induction on $n$. It is immediate to prove that point i) holds for $n = 0$. Next we prove that if point i) holds for $n$ then also point ii) holds for $n$. By inductive hypothesis there exist unique $\delta_0 : \ldots : \delta_{n-1}$, $\alpha'$ and $\beta'$ s.t.:
$D(\alpha, \beta) \Rightarrow \delta_0 : \ldots : \delta_{n-1} : D_1(\alpha', \beta')$ and $\alpha' \neq 1:1:\alpha'_{|_2}$, therefore it is possible to apply a (unique) rewriting rule and obtain: $D(\alpha, \beta) \Rightarrow \delta_0 : \ldots : \delta_{n-1} : D_2(\gamma'', \alpha'', \beta'')$. The proof that the restrictions on the values of $\alpha''$, $\beta''$ and $\gamma''$ hold is done by case analysis on the initial digits of the stream of $\alpha'$. The cases to consider are: $\alpha' = 0:0:\alpha'_{|_2}$, $\alpha' = 0:1:0:\alpha'_{|_3}$, $\alpha' = 0:1:1:\alpha'_{|_3}$ and $\alpha' = 1:0:\alpha'_{|_2}$. Here we consider the second and the third case only. The proofs for the remaining cases follow a similar pattern. If $\alpha' = 0:1:0:\alpha'_{|_3}$ then $D_1(\alpha', \beta') \Rightarrow D_2(A(0:\alpha'_{|_3}, 0:\beta', 1, 1), 1:0:\alpha'_{|_3}, \beta'))$. Let $\alpha'' = 1:0:\alpha'_{|_3}$, $\beta'' = \beta'$ and $\gamma'' = A(0:\alpha'_{|_3}, 0:\beta', 1, 1)$. Obviously $\alpha'' \neq 1:1:\alpha''_{|_2}$, moreover we have:

$$\begin{aligned}
[\![\gamma'']\!]_s &= [\![A(0:\alpha'_{|_3}, 0:\beta', 1, 1)]\!]_s \\
&= [\![\alpha'_{|_3}]\!]_s/\phi^3 + [\![\beta']\!]_s/\phi^3 + 1/\phi^3 + 1/\phi^4 \\
&= 1/\phi^2 + [\![\alpha'_{|_3}]\!]_s/\phi^3 + [\![\beta']\!]_s/\phi^3 \\
&= [\![\alpha']\!]_s + [\![\beta']\!]_s/\phi^3 \\
&= [\![\alpha']\!]_s + [\![\beta']\!]_s/\phi^2 - [\![\beta']\!]_s/\phi^4 \\
&\leq 2/\phi - [\![\beta']\!]_s/\phi^4 \qquad \text{(by inductive hypothesis)}
\end{aligned}$$

therefore: $[\![\gamma'']\!]_s + [\![\beta'']\!]_s/\phi^4 \leq 2/\phi$ and $[\![\gamma'']\!]_s = [\![\alpha'']\!]_s + [\![\beta'']\!]_s/\phi^3$.

We prove by absurdity that: $\gamma'' \neq 1:1:1:\gamma''_{|_3}$. If $\gamma'' = 1:1:1:\gamma''_{|_3}$ then $[\![\gamma'']\!]_s = 1/\phi + 1/\phi^2 + 1/\phi^3 + [\![\gamma''_{|_3}]\!]_s/\phi^3 = 2/\phi + [\![\gamma''_{|_3}]\!]_s/\phi^3$ and since $[\![\gamma'']\!]_s + [\![\beta'']\!]_s/\phi^4 \leq 2/\phi$, it follows that both the streams $\gamma''_{|_3}$ and $\beta$ have all elements equal to 0. A simple analysis on the algorithm $A$ shows that this in contradiction with the fact that: $\gamma'' = A(0:\alpha'_{|_3}, 0:\beta', 1, 1)$.

If $\alpha' = 0:1:1:\alpha'_{|_3}$ then $D_1(\alpha', \beta') \Rightarrow D_2(A(1:\alpha'_{|_3}, 0:\beta', 1, 1), 1:1:\alpha'_{|_3}, \beta')$
$\Rightarrow D_2(1 : A(\alpha'_{|_3}, \beta', 0, 0), 1:1:\alpha'_{|_3}, \beta')$ so in this case: $\gamma'' = 1 : A(\alpha'_{|_3}, \beta', 0, 0)$. Using the same arguments used for the previous case, it is possible to prove that: $\gamma'' \neq 1:1:1:\gamma''_{|_3}$ and $[\![\gamma'']\!]_s + [\![\beta'']\!]_s/\phi^4 \leq 2/\phi$.

Finally we prove that if point ii) holds for $n$ then the point $i$) holds for $n+1$. By inductive hypothesis there exist unique $\delta_0 : \ldots : \delta_{n-1}$, $\alpha'$, $\beta'$ s.t.:
$D(\alpha, \beta) \Rightarrow \delta_0 : \ldots : \delta_{n-1} : D_2(\gamma', \alpha', \beta')$. We reason by case analysis on the initial sub-stream of $\gamma'$. The cases to be considered are: $\gamma' = 0:0:\gamma'_{|_2}$, $\gamma' = 0:1:0:\gamma'_{|_3}$, $\gamma' = 0:1:1:\gamma'_{|_3}$, $\gamma' = 1:\gamma'_{|_1}$. Here we will consider the second and the fourth case only.

If $\gamma' = 0:1:0:\gamma'_{|_3}$ then $D(\alpha, \beta) \Rightarrow \delta_0 : \ldots : \delta_{n-1} : D_2(0:1:0:\gamma'_{|_3}, \alpha', \beta') \Rightarrow \delta_0 : \ldots : \delta_{n-1} : 0 : D_1(\alpha', \beta')$. By inductive hypothesis $\alpha' \neq 1 : 1 : \alpha'_{|_2}$ and: $[\![\alpha']\!]_s + [\![\beta']\!]_s/\phi^3 = [\![\gamma']\!]_s = 1/\phi^2 + [\![\gamma'_{|_3}]\!]_s/\phi^3 \leq 1/\phi^2 + 1/\phi^2 = 2/\phi^2$

If $\gamma' = 1:\gamma'_{|_1}$ then $D(\alpha, \beta) \Rightarrow \delta_0 : \ldots : \delta_{n-1} : D_2(1:\gamma'_{|_1}, \alpha', \beta') \Rightarrow \delta_0 : \ldots : \delta_{n-1} : 1 : D_1(\gamma'_{|_1}, \beta')$ By inductive hypothesis $\gamma'_{|_1} \neq 1:1:\gamma'_{|_3}$. Moreover we have:

$$\begin{aligned}
[\![\gamma'_{|_1}]\!]_s + [\![\beta']\!]_s/\phi^2 &= ([\![\gamma']\!]_s - 1/\phi) \times \phi + [\![\beta']\!]_s/\phi^2 \\
&\leq (2/\phi - [\![\beta']\!]_s/\phi^4 - 1/\phi) \times \phi + [\![\beta']\!]_s/\phi^2 \qquad \text{(by inductive hypothesis)} \\
&= 1 - [\![\beta']\!]_s/\phi^3 + [\![\beta']\!]_s/\phi^2 = 1 + [\![\beta']\!]_s/\phi^4 \\
&\leq 1 + 1/\phi^3 = 2/\phi.
\end{aligned}$$

□

To prove the correctness of the algorithm we need the following lemma.

**Lemma 10** *For every integer number n and for every binary streams $\alpha$, $\beta$ and $\gamma$:*
*i) if $D_1(\alpha, \beta)$ converges then:*

$$| [\![D_1(\alpha, \beta)]\!]_s - (\phi^2 \times [\![\alpha]\!]_s)/(\phi^2 - [\![\beta]\!]_s) | \leq \phi^{2-n},$$

*ii) if $D_2(\gamma, \alpha, \beta)$ converges and $\gamma = 1:\gamma_{|_1}$ then:*

$$| \llbracket D_2(\gamma, \alpha, \beta)\rrbracket_s - (\phi^3 \times \llbracket\gamma\rrbracket_s - \llbracket\beta\rrbracket_s)/(\phi^3 - \phi \times \llbracket\beta\rrbracket_s) | < \phi^{2-n},$$

*iii) if* $D_2(\gamma, \alpha, \beta)$ *converges and* $\llbracket\gamma\rrbracket_s = \llbracket\alpha\rrbracket_s/\phi + \llbracket\beta\rrbracket_s/\phi^3$ *then:*

$$| \llbracket D_2(\gamma, \alpha, \beta)\rrbracket_s - (\phi \times \llbracket\alpha\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s) | < \phi^{2-n},$$

**Proof.** The proof is by induction on the natural number $n$. A simple calculation shows that point i) holds for $n = 0$.

If point i) holds for $n$ then point ii) holds for $n+1$. In fact for $\gamma = 1{:}\gamma_{|_1}$ we have:

$$\begin{aligned}
&| \llbracket D_2(1{:}\gamma_{|_1}, \alpha, \beta)\rrbracket_s - (\phi^3 \times \llbracket 1{:}\gamma_{|_1}\rrbracket_s - \llbracket\beta\rrbracket_s)/(\phi^3 - \phi \times \llbracket\beta\rrbracket_s) | \\
&= | 1/\phi + \llbracket D_1(\gamma_{|_1}, \beta)\rrbracket_s/\phi - (\phi^2 + \phi^2 \times \llbracket\gamma_{|_1}\rrbracket_s - \llbracket\beta\rrbracket_s)/(\phi^3 - \phi \times \llbracket\beta\rrbracket_s) | \\
&= | 1/\phi + \llbracket D_1(\gamma_{|_1}, \beta)\rrbracket_s/\phi - 1/\phi - \phi^2 \times \llbracket\gamma_{|_1}\rrbracket_s/(\phi^3 - \phi \times \llbracket\beta\rrbracket_s) | \\
&= | \llbracket D_1(\gamma_{|_1}, \beta)\rrbracket_s - \phi^2 \times \llbracket\gamma_{|_1}\rrbracket_s/(\phi^2 - \llbracket\beta\rrbracket_s) | \ /\phi \\
&\leq \phi^{2-(n+1)} \qquad \text{(by inductive hypothesis)}
\end{aligned}$$

Next we prove, by case analysis on $\gamma$, that if point i) holds for $n$ then point iii) holds for $n+1$. The cases to consider are: $\gamma = 0{:}0{:}\gamma_{|_2}$, $\gamma = 0{:}1{:}0{:}\gamma_{|_3}$, $\gamma = 0{:}1{:}1{:}\gamma_{|_3}$, $\gamma = 1{:}\gamma_{|_1}$. Here we consider the second and the third case only.

If $\gamma = 0{:}1{:}0{:}\gamma_{|_3}$ then

$$\begin{aligned}
&| \llbracket D_2(0{:}1{:}0{:}\gamma'_{|_3}, \alpha, \beta)\rrbracket_s - (\phi \times \llbracket\alpha\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s) | \\
&= | \llbracket 0{:}D_1(\alpha, \beta)\rrbracket_s - (\phi \times \llbracket\alpha\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s) | \\
&= | D_1(\alpha, \beta)\rrbracket_s/\phi - (\phi \times \llbracket\alpha\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s) | \\
&\leq \phi^{2-(n+1)}
\end{aligned}$$

If $\gamma = 0{:}1{:}1{:}\gamma_{|_3}$ then:

$$\begin{aligned}
&| \llbracket D_2(0{:}1{:}1{:}\gamma'_{|_3}, \alpha, \beta)\rrbracket_s - (\phi \times \llbracket\alpha\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s) | \\
&= | \llbracket 1{:}D_1(0{:}0{:}\gamma'_{|_3}, \beta)\rrbracket_s - (\phi \times \llbracket\alpha\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s) | \\
&= | \llbracket 1{:}D_1(0{:}0{:}\gamma'_{|_3}, \beta)\rrbracket_s - (\phi^2 \times \llbracket 0{:}1{:}1{:}\gamma'_{|_3}\rrbracket_s - \llbracket\beta\rrbracket_s/\phi)/(\phi^2 - \llbracket\beta\rrbracket_s) | \\
&= | 1/\phi + \llbracket D_1(0{:}0{:}\gamma'_{|_3}, \beta)\rrbracket_s/\phi - (\phi + \phi \times \llbracket 0{:}0{:}\gamma'_{|_3}\rrbracket_s - \llbracket\beta\rrbracket_s/\phi)/(\phi^2 - \llbracket\beta\rrbracket_s) | \\
&= | \llbracket D_1(0{:}0{:}\gamma'_{|_3}, \beta)\rrbracket_s/\phi - \phi^2 \times \llbracket 0{:}0{:}\gamma'_{|_3}\rrbracket_s/(\phi^2 - \llbracket\beta\rrbracket_s) | \ /\phi \\
&\leq \phi^{2-(n+1)}
\end{aligned}$$

The last step consists in proving that if points ii) and iii) hold for $n$ then also point i) holds for $n$. The proof is by case analysis on the initial digits of $\alpha$. The cases to consider are: $\alpha = 0{:}0{:}\alpha_{|_2}$, $\alpha = 0{:}1{:}\alpha_{|_2}$, and $\alpha = 1{:}0{:}\alpha_{|_2}$. Here we consider the second and the third case only.

If $\alpha = 0{:}1{:}\alpha_{|_2}$ then

$$\begin{aligned}
&| \llbracket D_1(0{:}1{:}\alpha_{|_2}, \beta)\rrbracket_s - (\phi^2 \times \llbracket 0{:}1{:}\alpha_{|_2}\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s) | \\
&= | \llbracket D_2(A(\alpha_{|_2},\, 0{:}\beta,\, 1,\, 1), 1{:}\alpha_{|_2}, \beta)\rrbracket_s - (\phi \times \llbracket 1{:}\alpha_{|_2}\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s) | \\
&\leq \phi^{2-n} \qquad \text{, by inductive hypothesis since,} \\
&\llbracket A(\alpha_{|_2},\, 0{:}\beta,\, 1,\, 1)\rrbracket_s \quad = (\llbracket\alpha_{|_2}\rrbracket_s + \llbracket 0{:}\beta\rrbracket_s + 1/\phi + 1/\phi^2)/\phi^2 \\
&\qquad\qquad\qquad\qquad = 1/\phi^2 + \llbracket\alpha_{|_2}\rrbracket_s/\phi^2 + \llbracket\beta\rrbracket_s/\phi^3 \\
&\qquad\qquad\qquad\qquad = \llbracket 1{:}\alpha_{|_2}\rrbracket_s/\phi + \llbracket\beta\rrbracket_s/\phi^3
\end{aligned}$$

If $\alpha = 1{:}0{:}\alpha_{|_2}$ then

$$\begin{aligned}
&| \llbracket D_1(1{:}0{:}\alpha_{|_2}, \beta)\rrbracket_s - (\phi^2 \times \llbracket\alpha\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s) | \\
&= | \llbracket D_2(A(\alpha_{|_2},\, 1{:}\beta,\, 1,\, 1), \alpha_{|_2}, \beta)\rrbracket_s - (\phi^2 \times \alpha\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s) | \\
&= | \llbracket D_2(1{:}\gamma_{|_1}, \alpha_{|_2}, \beta)\rrbracket_s - (\phi^2 \times \llbracket\alpha\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s) |
\end{aligned}$$

in fact $A(\alpha_{|_2},\, 1{:}\beta,\, 1,\, 1)$ necessarily reduces to a stream of form $1{:}\gamma_{|_1}$, moreover we have:

$$\begin{aligned}\llbracket\gamma\rrbracket_s &= (\llbracket\alpha_{|_2}\rrbracket_s + \llbracket 1{:}\beta\rrbracket_s + 1/\phi + 1/\phi^2)/\phi^2\\ &= \llbracket\alpha_{|_2}\rrbracket_s/\phi^2 + 1/\phi^3 + \llbracket\beta\rrbracket_s/\phi^3 + 1/\phi^3 + 1/\phi^4\\ &= 1/\phi + \llbracket\alpha_{|_2}\rrbracket_s/\phi^2 + \llbracket\beta\rrbracket_s/\phi^3\\ &= \llbracket\alpha\rrbracket_s + \llbracket\beta\rrbracket_s/\phi^3\end{aligned}$$

so we can write:

$$\begin{aligned}&|\ \llbracket D_2(1{:}\gamma_{|_1}, \alpha_{|_2}, \beta)\rrbracket_s - (\phi^2 \times \llbracket\alpha\rrbracket_s)/(\phi^2 - \llbracket\beta\rrbracket_s)\ |\\ &= |\ \llbracket D_2(1{:}\gamma_{|_1}, \alpha_{|_2}, \beta)\rrbracket_s - (\phi^2 \times \llbracket\gamma\rrbracket_s - \llbracket\beta\rrbracket/\phi)/(\phi^2 - \llbracket\beta\rrbracket_s)\ |\\ &= |\ \llbracket D_2(1{:}\gamma_{|_1}, \alpha_{|_2}, \beta)\rrbracket_s - (\phi^3 \times \llbracket\gamma\rrbracket_s - \llbracket\beta\rrbracket)/(\phi^3 - \phi \times \llbracket\beta\rrbracket_s)\ |\\ &\leq \phi^{2-n}\end{aligned}$$

□

Using the above two lemmas one can easily prove the proposition. □

The algorithm for the division in the full notation is defined by:

**Definition 14** *The function $D'$ implementing the division in the full notation is defined by:*

$$\begin{array}{lcl} D'(z{:}\alpha,\ t{:}0{:}\beta) & \Rightarrow & D'_1(C'((z-t){:}\alpha),\ 0{:}C(\beta))\\ D'(z{:}\alpha,\ t{:}1{:}0{:}\beta) & \Rightarrow & D'(z{:}\alpha,\ (t-1){:}\beta)\\ D'(z{:}\alpha,\ t{:}1{:}1{:}\beta) & \Rightarrow & D'_1((z-t+1){:}\alpha,\ \beta)\\ D'_1(z{:}\alpha,\ 0{:}0{:}\beta) & \Rightarrow & D'_1((z+1){:}\alpha,\ \beta)\\ D'_1(z{:}\alpha,\ 0{:}1{:}\beta) & \Rightarrow & (z+1){:}D_1(A(\alpha,\ \beta,\ 0,\ 0),\ C(\beta))\\ D'_1(z{:}\alpha,\ 1{:}\beta) & \Rightarrow & (z+1){:}D_1(A(0{:}\alpha,\ \beta,\ 0,\ 1),\ C(\beta))\end{array}$$

**Proposition 11** *For every pair of integers $z$, $t$ and for every pair of streams $\alpha$, $\beta$ if $\llbracket t{:}\beta\rrbracket_f \neq 0$ then:*

$$\llbracket D'(z{:}\alpha,\ t{:}\beta)\rrbracket_f = \llbracket z{:}\alpha\rrbracket_f / \llbracket t{:}\beta\rrbracket_f$$

*If $\llbracket t{:}\beta\rrbracket_f = 0$ the evaluation of $D'(z{:}\alpha,\ t{:}\beta)$ diverges.*

**Proof.** The proposition follows from the fact that if $\llbracket\beta\rrbracket_s \neq 0$ then

$$\llbracket D'_1(z{:}\alpha,\ \beta)\rrbracket_f = \llbracket z{:}\alpha\rrbracket_f / \llbracket\beta\rrbracket_s.$$

From the proof of Lemma 9 it follows that the evaluation of $D'_1(z{:}\alpha,\ \beta)$ converges, if $\llbracket\beta\rrbracket_s \neq 0$. From Lemma 10 it follows that the function $D'_1$ is correct. The actual proof is omitted since it uses the same techniques presented in the previous proofs. □

## 5. Different base values

In this chapter we investigate further the idea of using a real number as base for a digit notation. In particular we consider the problem of whether there exist values, different from $\phi$, that can be used as bases for *binary* notations and which lead to simple algorithms for the arithmetic operations. We show that such values exist and we give a requirement they have to satisfy. It is interesting to notice that only irrational numbers satisfy this requirement.

In our analysis we make two simplifications. First we only consider the addition. This is justified because almost all algorithms for the other arithmetic operations are based on the one for additions. Subtraction can be obtained by complementing and adding. The algorithms we learned from primary school reduce multiplication to a series of additions, and division to a series of subtractions. Secondly we only consider a simplified notations for real numbers. An obvious generalisation of Definition 5 is the following:

**Definition 15** *In the simplified notations with base $x$ the binary stream $\alpha = \alpha_0 : \alpha_1 : \ldots$ denotes the real number:*

$$[\![\alpha]\!]_{s_x} = \sum_{i \in I\!N} \alpha_i \times x^{i+1}$$

The simplified notations are the common parts for all binary notations for real numbers independently from the methods used to represent arbitrary large real numbers.

The algorithm for addition of Definition 6 has a time complexity that is linear with the length of the arguments and a constant space complexity. The algorithm performs the addition reading the input, generating the output and using a limited amount of internal data structure. The following theorem states a necessary requirement for the base value in order to have algorithms for addition having constant space complexity.

**Proposition 12** *If, for the simplified notation with base $x$, there exist an algorithm for addition needing only a limited amount of internal data structure then there exist a natural number $n$ and a $n$-tuple of integers $c_0 \ldots c_{n-1}$ such that:*

$$x^n + c_{n-1}x^{n-1} + \ldots + c_0 = 0$$

*and $c_i \in \{1, 0, -1, -2, -3\}$ for all $i < n$.*

**Proof.** Since the algorithm uses just a limited amount of memory it has just a finite number of states. It follows that the algorithm generates a stream eventually periodic when it receives as input two eventually periodic streams. Let $l$ be a natural number such that having defined a stream $\alpha$ by $\alpha_i = 0$ for $i < l$ and $\alpha_i = 1$ for $i \geq l$ we have that the value $[\![\alpha]\!]_{s_x} + [\![\alpha]\!]_{s_x}$ can be represented in the simplified notation (any $l > \ln 2 / \ln x$ satisfies the condition). Let $\beta$ be the stream generated by the algorithm for addition when the input streams are both equal to $\alpha$. $\beta$ is eventually periodic, that is there exist $p$ and $q$ such that $\beta_i = \beta_{i+p}$ for all the $i > q$ . Let $m$ be the smallest natural s.t. $\beta_m = 1$, obviously $m < l$. The following equalities hold:

$$\sum_{i \geq l} 2/x^{i+1} = \sum_{i \geq 0} \beta_i/x^{i+1} = 1/x^{m+1} + \sum_{m < i \leq q} \beta_i/x^{i+1} + (\sum_{h \geq 0} 1/x^{hp})(\sum_{q < i \leq q+p} \beta_i/x^{i+1})$$

by evaluating the periodic series:

$$2/(x^l(x-1)) = 1/x^{m+1} + \sum_{m < i \leq q} \beta_i/x^{i+1} + x^p/(x^p-1)(\sum_{q < i \leq q+p} \beta_i/x^{i+1})$$

multiplying by $x^{l+q+1}(x^p - 1)$

$$2x^{q+1}(x^p-1)/(x-1) = x^{l+q-m}(x^p-1) + (\sum_{m < i \leq q} \beta_i x^{l+q-i})(x^p-1) + \sum_{q < i \leq q+p} \beta_i x^{p+l+q-i}$$

$$2\sum_{q < i \leq q+p} x^i = x^{p+q+l-m} + \sum_{m < i \leq q} \beta_i x^{p+q+l-i} - x^{q+l-m} - \sum_{m < i \leq q} \beta_i x^{q+l-i} + \sum_{q < i \leq q+p} \beta_i x^{q+p+l-i}$$

and from this the thesis. □

**Corollary 13** *If, for the simplified notation with base $x$, there exists an algorithm for addition needing only a limited amount of internal data structure then $x$ is not a rational number.*

**Proof.** A theorem of elementary algebra states that all the rational solutions of an equation of form $a_n y^n + \ldots + a_0 = 0$, with $a_i$ integer coefficients and $a_n \neq 0 \neq a_0$, can be written as $p/q$ where $p$ is integer dividing $a_0$ and $q$ is an integer dividing $a_n$. From this theorem it follows that the only possible rational values for the base $x$ are 1, 2 or 3. Between these values only 2 permits a binary representation of all the reals, but we know that with base 2 addition on reals is not computable. □

The golden ratio has been chosen as base in order to have the equality $1.00 = 0.11$. On the other hand Proposition 12 states that we can obtained simple algorithms for the addition only if the base is such that an equalities of the form $1.a_1 \ldots a_n = 0.b_1 \ldots b_n + 0.b'_1 \ldots b'_n + 0.b''_1 \ldots b''_n$, with $a_i, b_i, b'_i, b''_i \in \{0, 1\}$ holds. It is natural to consider the problem of whether there are bases different from $\phi$, which permit to write limited memory algorithms for the addition. It turns out that such bases exist. One example is the base for which the equality $1 = 0.111$ holds. In this case the base is the solution of the equation $x^3 - x^2 - x - 1 = 0$. This equation has only one real solution. We denote it by $\chi$. $\chi$ is a number greater than $\phi$ and smaller that 2. In the appendix we give the rewriting rules for the addition in base $\chi$. The rules have the same structure of the ones given in Definition 6 for base the golden ratio. But with base $\chi$ one needs a bigger set of rules: 10 rules for base $\phi$, 39 rules for base $\chi$. It is interesting to observe that in order to write the algorithm and prove its correctness we do not need to know the actual value of $\chi$ (which is still unknown to the author), we just exploit the equality $1 = 0.111$.

In the author opinion another base which probably permits a limited memory algorithm for the addition is the one for which the equality $1 = 0.101$ holds. It is still an open problem to define a criteria to establish which equalities lead to limited memory algorithms for the addition and which do not. An obvious requirement is the following: the base value generated by the equality has to be strictly bigger that 1 and strictly smaller that 2. In the family of equations defined in Proposition 12 there are four equations having degree strictly smaller than 3 and satisfying the above criteria. They are $x^2 - 2 = 0$, $x^2 - 3 = 0$, $x^2 + x - 3 = 0$, and $x^2 - x - 1 = 0$. The positive solutions of these equations are $\sqrt{2}$, $\sqrt{3}$, $(-1 + \sqrt{13})/2$, and $\phi$. It is possible to prove that there is no limited memory algorithm for addition when base is any of the values $\sqrt{2}$, $\sqrt{3}$, and $(-1 + \sqrt{13})/2$. Therefore all bases with a limited memory algorithm for addition and different from $\phi$ solve only equations having degree larger than 2. These equations are generated by equalities between binary strings involving four or more digits. In our experience the equalities that a base satisfied are fundamental tools in deriving algorithms for arithmetic operations. Since the golden ratio is the base satisfying the simplest equality, the golden ratio is probably also the base with the simplest algorithms for the arithmetic operations. This empirical argument is in some way sustained also by the complexity of the algorithm for addition in base $\chi$.

References

1. A. Avizienis. Binary-computable signed-digit arithmetic. In *AFIPS Conference Proceedings 26,1*, pages 663–672, 1964.
2. H.-J. Boehm. Constructive real interpretation of numerical programs. *SIGPLAN Notice*, 22, 7:214–221, July 87.
3. H.-J. Boehm and R. Cartwright. Exact real arithmetic: formulating real numbers as functions. In David Turner, editor, *Research topics in functional programming*, pages 43–64. Addison-Wesley, 1990.
4. L.E.J. Brouwer. Beweis, dass jede volle funktion gleichmässig stetig ist. In *Proc. Amsterdam 27*, pages 189–194, 1924.
5. A. Cauchy. Sur le moyens d'éviter les erreurs dans les calcules numériques. *Comptes Rendus*, 11:789–798, 1840. Republished in: Augustin Cauchy, Œvres complètes, 1ér série, Tome V, pp 431–442.
6. P. Di Gianantonio. *A functional approach to real number computation*. PhD thesis, University of

Pisa, 1993.

7. I. J. Holyer. *Functional programming with Miranda*. Pitman, London, repr. edition, 1993.
8. D. E. Knuth. *The art of computer programming.*, volume 2/Seminumerical algorithms. Addison-Wesley, 1969.
9. K. Ko. *Complexity theory of real functions*. Birkhauser, Boston, 1991.
10. P. Lanzi. Complessità degli algoritmi per l'aritmetica reale esatta. Master's thesis, Università di Udine, 1994.
11. J. Leslie. *The Philosophy of Arithmetic.* Edimburgh, 1817.
12. P. Martin-Löf. *Note on Constructive Mathematics.* Almqvist and Wiksell, Stockholm, 1970.
13. W. Parry. On the $\beta$-expansions of real numbers. *Acta Mathematica, Acad. Sci. Hung.*, 11:401–416, 1960.
14. Ph. Sünderhauf. A faithful computational model of the real numbers. Technical Report 1610, Fachbereich Matematik, University of Darmstadt, 1994.
15. A.M. Turing. On computable numbers, with an application to the entscheidungs problem. In *Proc. London Math. Soc. 42*, pages 230–265, 1937.
16. J. Vuillemin. Exact real computer arithmetic with continued fraction. In *Proc. A.C.M. conference on Lisp and functional Programming*, pages 14–27, 1988.
17. K. Weihrauch. *Computability*. Springer-Verlag, Berlin, Heidelberg, 1987.
18. E. Wiedmer. Computing with infinite objects. *Theoret. Comp. Sci.*, 10:133–155, 1980.

## 1. Addition rules for base $\chi$

**Definition 16** *The simplified algorithm for addition between in base $\chi$ is defined by the following set of rewriting rules:*

$$
\begin{array}{lcl}
A_\chi(0{:}\alpha,\ 0{:}\beta,\ 0,\ 0,\ c,\ c') & \Rightarrow & 0{:}A_\chi(\alpha,\ \beta,\ 0,\ c,\ c',\ 0)\\
A_\chi(1{:}\alpha,\ 0{:}\beta,\ 0,\ 0,\ 0,\ c) & \Rightarrow & 0{:}A_\chi(\alpha,\ \beta,\ 1,\ 0,\ c,\ 0)\\
A_\chi(1{:}\alpha,\ 0{:}\beta,\ 0,\ 0,\ c,\ 0) & \Rightarrow & 0{:}A_\chi(\alpha,\ \beta,\ 1,\ c,\ 0,\ 0)\\
A_\chi(1{:}1{:}1{:}\alpha,\ 0{:}1{:}1{:}\beta,\ 0,\ 0,\ 1,\ 1) & \Rightarrow & 1{:}0{:}0{:}A_\chi(\alpha,\ \beta,\ 0,\ 0,\ 1,\ 1)\\
A_\chi(1{:}0{:}\alpha,\ 1{:}0{:}\beta,\ 0,\ 0,\ 0,\ c) & \Rightarrow & 0{:}1{:}A_\chi(\alpha,\ \beta,\ 0,\ c,\ 1,\ 0)\\
A_\chi(1{:}1{:}\alpha,\ 1{:}0{:}\beta,\ 0,\ 0,\ 0,\ 0) & \Rightarrow & 0{:}1{:}A_\chi(\alpha,\ \beta,\ 1,\ 0,\ 1,\ 0)\\
A_\chi(1{:}1{:}1{:}\alpha,\ 1{:}0{:}1{:}\beta,\ 0,\ 0,\ 0,\ 1) & \Rightarrow & 1{:}1{:}0{:}A_\chi(\alpha,\ \beta,\ 0,\ 0,\ 0,\ 1)\\
A_\chi(1{:}1{:}0{:}\alpha,\ 1{:}1{:}0{:}\beta,\ 0,\ 0,\ 0,\ 0) & \Rightarrow & 0{:}1{:}1{:}A_\chi(\alpha,\ \beta,\ 0,\ 1,\ 0,\ 0)\\
A_\chi(1{:}1{:}1{:}\alpha,\ 1{:}1{:}\beta,\ 0,\ 0,\ 0,\ c) & \Rightarrow & 1{:}0{:}A_\chi(0{:}\alpha,\ \beta,\ 0,\ c,\ 0,\ 0)\\
A_\chi(1{:}1{:}\alpha,\ 1{:}1{:}\beta,\ 0,\ 0,\ 1,\ c) & \Rightarrow & 1{:}0{:}A_\chi(\alpha,\ \beta,\ 0,\ c,\ 1,\ 1)\\
A_\chi(1{:}0{:}a{:}\alpha,\ 1{:}0{:}0{:}\beta,\ 0,\ 1,\ 0,\ 0) & \Rightarrow & 0{:}1{:}1{:}A_\chi(\alpha,\ \beta,\ a,\ 1,\ 0,\ 0)\\
A_\chi(1{:}0{:}1{:}\alpha,\ 1{:}0{:}1{:}\beta,\ 0,\ 1,\ 0,\ 0) & \Rightarrow & 1{:}0{:}0{:}A_\chi(\alpha,\ \beta,\ 0,\ 1,\ 0,\ 1)\\
A_\chi(1{:}0{:}1{:}\alpha,\ 1{:}0{:}1{:}\beta,\ 0,\ 1,\ 0,\ 1) & \Rightarrow & 1{:}0{:}AA_\chi(\alpha,\ \beta,\ 1,\ 1,\ 0,\ 1)\\
A_\chi(1{:}1{:}\alpha,\ 1{:}\beta,\ 0,\ 1,\ c,\ c') & \Rightarrow & 1{:}A_\chi(0{:}\alpha,\ \beta,\ 0,\ c,\ c',\ 1)\\
A_\chi(0{:}a{:}\alpha,\ 0{:}0{:}\beta,\ 1,\ 0,\ 0,\ c) & \Rightarrow & 0{:}1{:}A_\chi(\alpha,\ \beta,\ a,\ c,\ 0,\ 0)\\
A_\chi(0{:}1{:}a{:}\alpha,\ 0{:}1{:}0{:}\beta,\ 1,\ 0,\ 0,\ 0) & \Rightarrow & 0{:}1{:}1{:}A_\chi(\alpha,\ \beta,\ a,\ 0,\ 1,\ 0)\\
A_\chi(0{:}1{:}1{:}\alpha,\ 0{:}1{:}1{:}\beta,\ 1,\ 0,\ 0,\ c) & \Rightarrow & 1{:}0{:}0{:}A_\chi(\alpha,\ \beta,\ c,\ 0,\ 1,\ 1)\\
A_\chi(0{:}1{:}\alpha,\ 0{:}1{:}\beta,\ 1,\ 0,\ 1,\ c) & \Rightarrow & 1{:}0{:}A_\chi(\alpha,\ \beta,\ 0,\ c,\ 0,\ 1)\\
A_\chi(1{:}0{:}a{:}\alpha,\ 0{:}0{:}0{:}\beta,\ 1,\ 0,\ 0,\ 0) & \Rightarrow & 0{:}1{:}1{:}A_\chi(\alpha,\ \beta,\ a,\ 0,\ 0,\ 0)\\
A_\chi(1{:}0{:}1{:}\alpha,\ 0{:}0{:}1{:}\beta,\ 1,\ 0,\ 0,\ c) & \Rightarrow & 1{:}0{:}0{:}A_\chi(\alpha,\ \beta,\ c,\ 0,\ 0,\ 1)\\
A_\chi(1{:}1{:}\alpha,\ 0{:}\beta,\ 1,\ 0,\ c,\ c') & \Rightarrow & 1{:}A_\chi(0{:}\alpha,\ \beta,\ 0,\ c,\ c',\ 0)\\
A_\chi(1{:}\alpha,\ 1{:}\beta,\ 1,\ 0,\ c,\ 0) & \Rightarrow & 1{:}A_\chi(\alpha,\ \beta,\ 0,\ c,\ 1,\ 1)\\
A_\chi(1{:}1{:}\alpha,\ 1{:}\beta,\ 1,\ 0,\ c,\ c') & \Rightarrow & 1{:}A_\chi(0{:}\alpha,\ \beta,\ 1,\ c,\ c',\ 0)
\end{array}
$$

$$
\begin{array}{lcl}
A_\chi(1{:}0{:}1{:}\alpha,\ 1{:}0{:}1{:}\beta,\ 1,\ 0,\ 0,\ 1) & \Rightarrow & 1{:}0{:}1{:}A_\chi(\alpha,\ \beta,\ 1,\ 0,\ 0,\ 1) \\
A_\chi(1{:}\alpha,\ 1{:}\beta,\ 1,\ 1,\ 0,\ 0) & \Rightarrow & 1{:}A_\chi(\alpha,\ \beta,\ 1,\ 0,\ 1,\ 1) \\
AA_\chi(\alpha,\ 0{:}\beta,\ 1,\ 1,\ 0,\ 1) & \Rightarrow & 0{:}A_\chi(\alpha,\ 1{:}\beta,\ 1,\ 0,\ 0,\ 1) \\
AA_\chi(1{:}1{:}\alpha,\ 1{:}1{:}\beta,\ 1,\ 1,\ 0,\ 1) & \Rightarrow & 1{:}0{:}0{:}A_\chi(\alpha,\ \beta,\ 0,\ 0,\ 0,\ 0) \\
AA_\chi(1{:}1{:}\alpha,\ 1{:}0{:}\beta,\ 1,\ 1,\ 0,\ 1) & \Rightarrow & 0{:}1{:}1{:}A_\chi(\alpha,\ \beta,\ 0,\ 1,\ 1,\ 0) \\
AA_\chi(1{:}0{:}\alpha,\ 1{:}0{:}0{:}\beta,\ 1,\ 1,\ 0,\ 1) & \Rightarrow & 0{:}A_\chi(1{:}0{:}\alpha,\ 1{:}0{:}1{:}\beta,\ 1,\ 1,\ 0,\ 0) \\
AA_\chi(1{:}0{:}1{:}\alpha,\ 1{:}0{:}1{:}\beta,\ 1,\ 1,\ 0,\ 1) & \Rightarrow & 0{:}1{:}1{:}AA_\chi(\alpha,\ \beta,\ 1,\ 1,\ 0,\ 1)
\end{array}
$$

$$
\begin{array}{lcl}
A_\chi(0{:}\alpha,\ 1{:}\beta,\ c_0,\ c_1,\ c_2,\ c_3) & \Rightarrow & A_\chi(1{:}\alpha,\ 0{:}\beta,\ c_0,\ c_1,\ c_2,\ c_3) \\
A_\chi(\alpha,\ 0{:}\beta,\ c_0,\ 1,\ c_2,\ c_3) & \Rightarrow & A_\chi(\alpha,\ 1{:}\beta,\ c_0,\ 0,\ c_2,\ c_3) \\
A_\chi(a_1{:}0{:}\alpha,\ b_1{:}1{:}\beta,\ c_0,\ c_1,\ c_2,\ c_3) & \Rightarrow & A_\chi(a_1{:}1{:}\alpha,\ b_1{:}0{:}\beta,\ c_0,\ c_1,\ c_2,\ c_3) \\
A_\chi(\alpha,\ b_1{:}0{:}\beta,\ c_0,\ c_1,\ 1,\ c_3) & \Rightarrow & A_\chi(\alpha,\ b_1{:}1{:}\beta,\ c_0,\ c_1,\ 0,\ c_3) \\
A_\chi(a_1{:}a_2{:}0{:}\alpha,\ b_1{:}b_2{:}1{:}\beta,\ c_0,\ c_1,\ c_2,\ c_3) & \Rightarrow & A_\chi(a_1{:}a_2{:}1{:}\alpha,\ b_1{:}b_2{:}0{:}\beta,\ c_0,\ c_1,\ c_2,\ c_3) \\
A_\chi(\alpha,\ b_1{:}b_2{:}0{:}\beta,\ c_0,\ c_1,\ c_2,\ 1) & \Rightarrow & A_\chi(\alpha,\ b_1{:}b_2{:}1{:}\beta,\ c_0,\ c_1,\ c_2,\ 0) \\
AA_\chi(0{:}\alpha,\ 1{:}\beta,\ 1,\ 1,\ 0,\ 1) & \Rightarrow & AA_\chi(1{:}\alpha,\ 0{:}\beta,\ 1,\ 1,\ 0,\ 1) \\
AA_\chi(a_1{:}0{:}\alpha,\ b_1{:}1{:}\beta,\ 1,\ 1,\ 0,\ 1) & \Rightarrow & AA_\chi(a_1{:}1{:}\alpha,\ b_1{:}0{:}\beta,\ 1,\ 1,\ 0,\ 1) \\
AA_\chi(a_1{:}a_2{:}0{:}\alpha,\ b_1{:}b_2{:}1{:}\beta,\ 1,\ 1,\ 0,\ 1) & \Rightarrow & AA_\chi(a_1{:}a_2{:}1{:}\alpha,\ b_1{:}b_2{:}0{:}\beta,\ 1,\ 1,\ 0,\ 1)
\end{array}
$$

**Proposition 14** *i) For every pair of streams $\alpha, \beta$ for every quadruple of binary digits, $c_0, c_1, c_2, c_3$ if $c_0 = 0$ or $c_1 = 0$ or $c_3 = c_4 = 0$ then there exists a stream $\gamma$ s. t.: $A_\chi(\alpha,\ \beta,\ c_0,\ c_1,\ c_2,\ c_3) \Rightarrow \gamma$ and we have:*

$$[\![\gamma]\!]_{s_\chi} = ([\![\alpha]\!]_{s_\chi} + [\![\beta]\!]_{s_\chi} + c_0 + c_1/\chi + c_2/\chi^2 + c_3/\chi^3)/\chi^2$$

*ii) For every pair of streams $\alpha, \beta$ there exists a stream $\gamma$ s. t.: $AA_\chi(\alpha, \beta, 1,\ 1,\ 0,\ 1) \Rightarrow \gamma$ and for any such a stream we have:*

$$[\![\gamma]\!]_{s_\chi} = ([\![\alpha]\!]_{s_\chi} + [\![\beta]\!]_{s_\chi} + 1 + 1/\chi + 1/\chi^3)/\chi^3$$