Division by Zero in Logic and Computing

Jan A Bergstra

Division by Zero in Logic and Computing

Jan A. Bergstra
Minstroom Research BV, Utrecht, The Netherlands
email: janaldertb@gmail.com

Submission for the first issue of "Division by Zero Calculus"
a new Journal initiated and edited by S. Saitoh.

April 1, 2021

Abstract

The phenomenon of division by zero is considered from the perspectives of logic and informatics respectively. Division rather than multiplicative inverse is taken as the point of departure. A classification of views on division by zero is proposed: principled, physics based principled, quasi-principled, curiosity driven, pragmatic, and ad hoc. A survey is provided of different perspectives on the value of $1/0$ with for each view an assessment view from the perspectives of logic and computing. No attempt is made to survey the long and diverse history of the subject.

Contents

1 Introduction

In the context of rational numbers the constants 0 and 1 and the operations of addition $(\_ + \_)$ subtraction $(\_ - \_)$ as well as multiplication $(\_ \cdot \_)$ and division $(\_/\_)$ play a key role. When starting with a binary primitive for subtraction unary opposite is an abbreviation as follows: $-x = 0 - x$, and given a two-place division function unary inverse is an abbreviation as follows: $x^{-1} = 1/x$.

Number systems may be developed in many ways. I will assume that a set $\mathbb{N}$ has been chosen which serves as the natural numbers including 0 and 1 and equipped with addition and multiplication. Now one may first contemplate subtraction. For many inputs from $\mathbb{N}$ subtraction is unproblematic, e.g. $7-5 = 2$ and subtraction is a valid operation by all means. However it turns out that say $5 - 7$ poses a problem. There seems to be a general consensus that the phenomenon of $5 - 7$ having no value in $\mathbb{N}$ is best solved by working in $\mathbb{Z}$, chosen as an extension of $\mathbb{N}$, containing negative values. Then $5 - 7 = -2$ and -2 is taken to be a value not in need of further evaluation.

Now in logic there is a significant tradition of taking the value of $7 - 5$ equal to 0 because one insists not to extend the domain $\mathbb{N}$. $\mathbb{N}$ has been brought to prominence in the second half of the 19th Century mainly by Dedekind and Peano. Gödel's results regarding the incompleteness of axioms for arithmetic, and for that reason for mathematics at large, are stated and proven about $\mathbb{N}$. In logic the tradition has grown to denote subtraction on natural numbers which returns 0 rather than negative values (which don't exist in the naturals) as monus $(\_ \dot{-} \_)$ so that $5 \dot{-} 7 = 0$. When asked about the true value of $5 - 7$ in the setting of natural numbers logicians are unimpressed and will reply that they prefer to use $\_ - \_$ in the context of integers where that question has a unique and convincing answer.

In informatics most (if not all) program notations will have $5 - 7$ well defined and equal to -2. In program notations it is uncommon to have a standard type `natural` which denotes a subtype of `integer` and the need for a monus function does not arise. Unsigned integers are common in program notations but these are conceived as finite domains and involve modulo arithmetic modulo the number of representable numbers (+1), so that working with monus is an unnecessary complication. The monus function has ample application in theoretical work, however.

Proceeding with division it is plausible to start out from $\mathbb{Z}$ rather than from $\mathbb{N}$ and consider the "problem of subtraction" solved, by having subtraction as

a total function, i.e. defined on all inputs. Division is clear for many inputs for instance $4/2 = 2$, but not for all inputs, for instance not for $3/2$. By extending the $\mathbb{Z}$ once more and embedding it in $\mathbb{Q}$, a field of rational numbers, most divisions become well-defined, though exceptional cases arise from attempts to divide by zero. This observation leads to the paradigmatic question: what is $1/0$? There is a long history of speculation and contemplation concerning this particular question. Today's conventional response, at least for the automated arithmetic of floating point approximations of real numbers, is that $1/0 = +\infty$, an idea which was implemented already by Konrad Zuse in the earliest stage of electronic computer design. The idea that $1/0 = +\infty$ is best appreciated as one option out of a plurality of views on division by zero, and mapping out the plurality of views on that matter is my objective in this paper. I will restrict focus mainly to considerations from logic and computing, leaving al but untouched the aspects (if any exist) of relevance for the identification of $1/0$ which have a background mathematics, physics, and philosophy.

Abbreviation: DbZ. Below I will use DbZ as an abbreviation for the theme of division by zero, not just for questions regarding the value and status of $1/0$.

1.1 Structure of the paper

The structure of the paper is as follows: first I will survey different ways in which DbZ may appear in logic and computer science. The survey takes the form of a 5 layer model, mainly focusing on computing. Thereafter I provide a list of qualifications of perspectives on DbZ. DbZ being a controversial topic such qualifications matter a lot, and it is helpful to have these qualifications in an explicit form. Then I will describe the current mainstream view on DbZ as it appears in educational practice as well as in most work of professional mathematicians. I will conclude that justification of the mainstream position is puzzling, an observation which may be contrasted with ubiquitous claims to the contrary.

Having given a detailed description I am in the position to survey alternative views of which there are many. I will distinguish 11 different alternative positions on DbZ. Although the positions on DbZ in the survey are theoretical positions, it is useful to contrast these positions with observations made on the handling of DbZ in different programming languages. Throughout the paper there will be links to programming practice in various program notations. The paper ends with some concluding remarks, phrased as conclusions about DbZ.

2 Aspects of division by zero in computing

A major source of information regarding questions about division by zero in the context of informatics has come about from programming languages and language design. This takes place at different layers.

2.1 Five layer model for DbZ in computer programming

I will speak of DbZ for any occurrence of division by zero. I will speak of DvZ handling in order to refer to the policy of dealing with DbZ for a program or program notation. DbZ (and DbZ handling) can be static or dynamic. For logic there is only the layer of theoretical reflection while for computing the situation is far more complicated than that. Five layers of abstraction concerning DbZ can be distinguished as follows.

Theory of computer science layer. In the top layer design proposals are contemplated and analysed, some of which may become ubiquitous and some of which may never be implemented.

Program notation independent standardisation layer. Program notation design is guided by various standards which are supposedly valid for a plurality of such design. A famous example is ASCII, an original standard for the binary representation of characters. In connection with DbZ the IEEE 754 floating point standard provides suggestions for how to design program notations when it comes to implementations of reals (in fact always a subset of rationals). This standard deals extensively with DbZ and, importantly, does not impose that a run is aborted as soon as DbZ occurs. On the contrary it is suggested to proceed calculations with a signed infinite value, the sign depending on the sign of the numerator. Dividing 0 by 0 is considered worse, but again instead of imposing abortion it is suggested that calculations proceed with `NaN` an entity.

The world of `NaN`s is complicated. There will be a plurality of `NaN`s, at least two of them. Silent `NaN`s are distinguished from signalling `NaN`s which cause an exception or an abort of the run.

Program notation specific standardisation layer. A design for a program notation requires a precise description (standard) which explains those who intend to implement a program notation what must be doen, and which explains users of such implementations what they may expect. A particular program notation may be documented ins such a manner that it is undefined (red unspecified) how to handle an instance of DbZ. The documentation may be much more specific about that matter as well.

Program notation implementation specific layer. This layer is amenable to empirical observation. Given a specific implementation of a program notation it can be analysed what behaviour results from programs the execution of which will contain instances of DbZ. The following aspects arise:

- Compile time rejection of a candidate program because the occurrence of DbZ follows from the text at hand. (This option is unfeasible as it is for a given text undecidable whether or not a fracterm with denominator equal to zero will occur during a computation. What could be done is to ask the programmer to prove that property and

to make the compiler check the proof. Admittedly forcing the programmer to work in that manner would be a very big step, which given current technology is not likely to be taken in the near future.)

- Run time abortion with an error message upon an occurrence of DbZ.
- Run time exception handling with an error message upon an occurrence of DbZ.
- Run time evaluation of DbZ into a specific non-numerical value (to name some options `NaN, inf, +inf,-inf`).
- Run time exception handling with an error message upon calculations following an occurrence of DbZ, the first steps of which have been handled with the help of non-numerical values.

Program specific layer. In the program specific layer it is analysed how and why in a particular application a certain policy on DbZ handling is made use of, and of course, if and how that use could be improved.

3 Qualitative labelling of positions on DbZ

Principled position. A principled position on DbZ involves the perspective that not only the position is consistent and workable but in addition the idea that as a position it is superior to alternative positions on DbZ, this superiority being based on quite general considerations.

Examples.

(i) The mainstream position on DbZ (strictly maintaining that an expression p/q is only used in a context where arguably q is nonzero).

(i) The Division by Zero calculus due to S. Saitoh et.al. (see e.g. [34] and references cited in that paper), adopting $\forall x(x/0 = 0)$ and the logical consequences thereof as a point of departure for developing arithmetic as well as analysis in mathematics).

(iii) The view that the equation $\forall x(x/x = 1)$ expresses a universally valid principle, and by consequence the assertion that $0/0 = 1$.

Physics based principled position. Some positions take inspiration from observations on calculations in physics and argue for the validity and significance of certain design decisions by pointing to correspondences with such computations.

Example.

The position that $0/0 = 1$ corresponds well with certain laws and calculations in physics.

Quasi-principled position. A quasi-principled position on DbZ involves the idea that at least for certain areas of application the position at hand is

arguably superior to alternative positions. A quasi-principled position on DbZ can accommodate without hesitation the existence of deviating alternative quasi-principled positions as long as these are meant for application in other areas.

Examples.

(i) The design of transrationals, transreals and transcomplex numbers ([2, 26]) reflects a quasi-principled position. This position is a realization (though not a unique one) in the context of DbZ of the more general principled position, held by the designers, of transmathematics, that mathematical systems are best equipped with total functions.

(ii) the common meadows of [14] (with $1/0 = \bot$) represent a quasi-principled position. Common meadows are suitable, more so than any of the other alternative views on DbZ, for formalizing the mainstream position on DbZ.

(iii) The IEEE 754 standard expresses a view (with $1/0 = +\infty$, and $0/0 = \Phi$, where Φ is Nullity, a transrational constant) which is principled in the sense that design decisions have been taken after ample reflection in a plurality or committees only and which is quasi-principled because the standard is a man-made entity the design of which always comes with possible but rejected alternatives. (The description of the requirements of the standard with $1/0 = +\infty$ and $0/0 = \Phi$ ought to be explained in terms of the somewhat counterintuitive "equational" logic of `NaN`s, but such details are better covered elsewhere.)

Curiosity driven position. A curiosity driven position is detached from any judgment of practical or theoretical value of the position at hand.

Example.

The work by Komori in 1975 ([31]) and by Ono in 1983 ([**?**]) on pseudofields, adopting $1/0 = 0$, both represent a curiosity driven position.

Pragmatic position. A pragmatic position involves a choice for a particular view on DbZ while acknowledging that for the application at hand the advantage of the chosen positions over some of the other possible positions on DbZ is at best marginal.

Example.

For instance choosing $1/0 = 0$ in the design of a proof checking system (e.g. Coq or Isabelle), is only marginally better (more convenient) than choosing $1/0 = 1$ or say $1/0 = 754$.

Ad hoc position. Ad ad hoc position on DbZ (or on a DbZ related matter) is a mere design decision without a clearly pronounced rationale.

Example.

As an example I mention the original design decision that $0/0 = 1$ in APL. (Not the case, however, in the online APL environment I have been experimenting with, see 7.1 below.)

4 Mainstream position on DbZ

According to mainstream views $1/0$ is undefined, and expressions like $1/0$ must be avoided as a matter of principle.

Adopting that $1/0$ has no definition, in other words that division is a partial function, in combination with pursuing and demanding a style of writing which avoids the use of any expression which is undefined for the reason of involving an occurrence of division by zero is the mainstream perspective on division by zero in mathematics and in school arithmetic as well.

The mainstream view will reject the assertion "$1/0$ is undefined" because any use of an expression like $1/0$ stands in sharp contrast with the supposedly well-known objective of dividing 1 by 0. Any attempt to find a value for $1/0$ is considered incoherent given the fact that $0 \cdot a = 0$ for all a and the existence of an a such that $0 \cdot a = 1$ can be excluded beforehand. But this observation must not be read as $1/0$ denoting an irrational number, as $1/0$ is far less rational than being merely irrational (i.e. not being the value of a ratio).

Following the remark made above in the first item of the description of the program specific standardisation layer, it is hardly conceivable to extend the mainstream conventions to the current practice of imperative programming. Doing so requires formal methods, programmer written correctness proofs, and a proof checking compilers, which is possible in principle, but not yet in practice. Having said that program verification technology is becoming better and better and may pessimism on the matter may soon be outdated.

4.1 Connections of the mainstream position with informatics and logic

- Most, if not all, work in theoretical informatics adopts the mainstream position on division by zero, unless the theme of division by zero is the topic at hand (I am unaware of any exception).

- Most, if not all, work in logic adopts the mainstream position on division by zero, unless the theme of division by zero is the topic at hand. However, in model theory it is often assumed that $1/0$ has some predefined (but arbitrary) value and the work is then arranged in such a manner that the choice of that value is immaterial.

- The status of division by zero is classically not regarded as a worry for the foundations of mathematics which is worth more than casual attention.

4.2 Elaboration of the mainstream position.

I notice that from the mainstream perspective the assertion that "$\sum_{n=1}^{\infty} 1/n$ is undefined" is adopted as a meaningful assertion. That is done in view of that fact that non-convergence of said infinite sum is a non-trivial mathematical insight itself worth of a mathematical proof. In fact the mainstream approach tolerates expressions without value in most, if not all, circumstances, while the mainstream approach rejects the use of expressions for which the nonexistence of a value is manifest without proof. Arguably division by zero is unique in that respect, and in all other cases a more flexible approach to the acceptance of notations is taken.

I hold that in the mainstream view "a result of dividing 1 by 0 cannot exist" represents the paradigmatic instance of mathematical non-existence. Moreover this very state of affairs is appreciated for its power to support the awareness of non-existence as a first class citizen in mathematical discourse at a very early level of mathematical education. Awareness of non-existence supports an understanding of mathematics as a battle against the (collective) enemy of non-existence.

4.3 Elusive foundations of the mainstream position

Because of the notorious ambiguity of "fraction", is it a value or is it an expression, I will speak of a fracterm if an expression is meant. A fracterm is an expression with division as the leading function symbol, I refer to [7] for an exposition on fracterms.

The "mainstream" objection against using the fracterm $1/0$ is founded on wide-spread certainty regarding validity of the equation $0 \cdot x = 0$ which implies $0 \cdot x \neq 1$. If, however, it is 100% certain that $0 \cdot x \neq 1$ then $1/0$ must be a non-denoting fracterm, i.e. a fracterm to which no meaning is assigned. And it is a plausible rule of thumb that surely non-denoting fracterms should not be used.

Now, the objection that $1/0$ does not exist because there is no known rational number p which satisfies $0 \cdot p = 1$ cannot be held against the use of $1/0$. Indeed there is no principled objection against the use of $\sqrt{2}$, an objection which is merely based on the absence of a rational number that satisfies $p^2 = 1$. And there is no principled objection against the use of $\sqrt{-1}$ based on the absence of a real number p that satisfies $p^2 = -1$. Notations are not rejected if these are meaningful in larger number systems. Now returning to the fracterm $1/0$ it is far from obvious why it is so often supposed to be self-evident that $x \cdot 0 = 1$ will not be satisfied in any significant extensions of the number system. If that is a theorem, it is in need of a proof. If it is an axiom then that should be said, if it is an empirical observation then the relevant facts must be put on the table.

Rejecting as mathematically non-professional and as a sign of defective understanding any use of the fracterm $1/0$ can only be based on adopting the truth of $0 \cdot x = 0$ (and of $0 \cdot x \neq 1$) as more far reaching than most "ordinary" axioms of arithmetic. Doing so can be done for different reasons, however, and at least

the following options for motivating that assumption may be distinguished, each of which support current mainstream notational practices regarding division by zero.

(i) $0 \cdot x = 0$ is considered a fundamental axiom which will hold forever by way of a collective mental decision, (and $0 \cdot x = 0$ is a more far reaching truth than for instance $x^2 \geq 0$ a fact which easily follows from the understanding of a square as the surface of a geometrical figure. It is also a more far reaching truth than $1 + 1 \neq 0$ which turns out to hold in a field of characteristic 2.)

(ii) Until now no mathematician has developed an extension of the known number system(s) with elements for which $0 \cdot x = 0$ fails and where $1/0$ acquires a proper value, and which has gained broad acceptance. At the same time one is aware that his state of affairs might conceivably change in the future. For instance the introduction of non-standard numbers in nonstandard analysis, in the 60ties of the 20th Century, has created new numbers. The introduction of nonstandard analysis has turned the Archimedian axiom $\forall x \in \mathbb{R}(x > 0 \rightarrow \bigvee_{n\in\mathbb{N}}(x < n))$ into a defeasible fact.

In other words: rejection of the use of $1/0$ is sufficiently grounded in the observation that no currently accepted number system introduces a value for that fracterm.

(iii) It is a fundamental but unprovable hypothesis underlying the design of mainstream arithmetic that no such extension of the known number systems will be found and for that reason any use of $1/0$ may be rejected. This hypothesis may be compared with the hypothesis that Peano's axioms are consistent.

(iv) A general dislike of the fracterm $1/0$ is a mainstream attitude at the moment but that observation must not be taken too seriously and mainstream thinking may change on this matter and may turn to alternative options in the future. That mathematical education nowadays pays much attention to integer numbers with addition subtraction and multiplication and to the extension to the field of rational numbers is perhaps only a temporary phase, and once other parts of mathematics gain prominence the focus on disputing the credibility of say $1/0$ may fade away.

In school arithmetic, students are often told that obviously there is no x such that $0 \cdot x = 1$, with as an immediate consequence that $1/0$ is obviously non-denoting. But I see no way in which it is (or might be) obvious (for beginning students) that for all plausible extensions of the integers, now and in the future, there will be no such x. once more contemplating $\sqrt{-1}$, which asks for a scenario where x^2 is negative, it must be acknowledged that a considerable amount of out of the box thinking may be needed to find a scenario where for some x, $0 \cdot x = 1$ holds.

4.4 Mismatch with terminology from informatics

In informatics, in particular in the science of computer programming, if a language standard states that say the result of dividing by zero is undefined that means that in an implementation anything can happen. If say it is required instead that upon having been instructed to divide by zero the run of a pro-

gram must abort with a runtime error message then that requirement must be explicitly stated.

In the theory of computation a value being undefined is often identified with divergence (i.e. non-termination) of the computation of that value (by a given program on a given machine or machine model). The latter interpretation of "undefined" has little connection with the interpretation of assertions that "division by zero is undefined".

In mathematics, however, the assertion that division by zero is undefined implies that an expression $1/0$ must not be given a meaning (even if that could be done).

4.5 Ammended mainstream: $0/0 = 0$ with $1/0$ undefined

The mainstream position may be somewhat more tolerant by adopting the fracterm $0/0$, and assuming that $0/0 = 0$. This assumption is consistent with the idea that when $x/y = z$ it must be the case that $y \cdot z = x$. Having $1/0$ undefined may go hand in hand with adopting $0/0 = 0$. I will refer to that position as the ammended mainstream position on DbZ.

4.6 Peripheral numbers: thinking outside the box

When deviation from the mainstream position it is plausible to think of "new" entities as numbers, and to allow there new entities not to comply with one or more conventional rules. I will speak of peripheral numbers and I will use a formal definition of that notion. The idea, though not formally required is that non-peripheral numbers constitute an additive ring, the peripheral numbers lying outside that ring.

Definition 4.1. *(Peripheral numbers) Let Σ contain $\_ + \_, -\_, 0$, all for sort s. An element p of sort A_s of a Σ-algebra A is peripheral in A if $p + (-p) \neq 0$.*

5 Grounds for deviating from the mainstream position on DbZ

Whoever deviates from the mainstream position will need to find reasons for doing so. In the case of DbZ there is a plurality of reasons one may put forward for motivating other positions. It is impossible to give an ordering of importance for these reasons because different alternative views differ on the matter of relative weight of different possible reasons.

Empirical grounds I. DbZ is a phenomenon which occurs or may occur during the run of a computer program. Even if considered undesirable the question how to proceed when a computation calls for the evaluation of an expression like $1/0$ cannot be avoided. Now the observation is that many (imperative) program notations will require that the fragment

$$\texttt{x} = \texttt{1/0}; \texttt{print(x)}; \texttt{print(',')}; \texttt{y} = -\texttt{x}; \texttt{print(y)}$$

(with `x` = `t` denoting an assignment, and `x` a variable capable of containing floating point values) will produces something like

$$+\texttt{Inf}\ ,\ -\texttt{Inf}$$

The key observation is that $1/0$ is supposed to have a signed value different from the usual values. This arrangement is so wide-spread that it is far from obvious that the mainstream position is worth its name. In the world of computer programming the mainstream position is a minority position.

Remark. I have not found any research work which explains in terms of examples and results about such examples why it is or might be an advantage that a run of a program proceeds, while working with "infinite" values instead of aborting or raising an exception. Of course there is empirical evidence that the use of peripheral values allows computing to proceed where it otherwise might have to stop, and that in this manner the use of peripheral values may be time-saving, but there seems to be no theoretical account of such matters which formulates said advantage in terms of stated theorems with proofs.

Abortion or raising an exception will happen in some but not all program notations with the following fragment:

$$\texttt{x} = \texttt{1/0}; \texttt{print(x)}; \texttt{print(}',\texttt{')}; \texttt{y} = \texttt{x} - \texttt{x}; \texttt{print(y)}$$

The idea is that although one may still claim that upon the assignment $\texttt{x} = \texttt{1/0}$ it is known that, say $\texttt{x} > \texttt{17}$, following the assignment $\texttt{y} = \texttt{x} - \texttt{x}$ no such information exists about `y` there is no reliable information about the size of `y` relative to ordinary values.

In any case from the perspective of current program notations: (i) an expression `1/0` is program text will in most notations not give rise to a compile time error, and (ii) it is the rule rather than the exception that `1/0` is given a value which can be used in computations and which more or less behaves like other (conventional) values.

Empirical grounds II. For computer supported proof systems it is conventional to adopt a rule of the form $1/0 = a$ for some number a and then to make the logic independent of the choice of a. A well-known choice for a is $a = 0$. In model theory (a branch of mathematical logic) a similar convention is often assumed, and in model theory it is usual to simply state that some fixed but unspecified a has been chosen for that purpose.

Pragmatic grounds I. If one prefers to use equational logic rather than first order logic it becomes a significant advantage to have all operations total. All known logics of partial functions (of which there are many) are quite complicated in comparison with the equational fragment of first order logic.

For instance if one intends to specify division by means of equational axioms, an ambition which is implicit in the methodology of datatypes and

abstract datatypes and which has its roots in software engineering, then working with total functions (and with totalised division in particular) turns out to be a significant advantage.

For fans of equational logic $1/0 = 0$ is a relevant choice because it gives rise to nice equations, to a workable logic and to an attractive meta-theory. The choice $1/0 = \bot$ is an alternative which, though less elegant, lies much closer to the mainstream ambition.

Pragmatic grounds II. One may hold that having a theory of numbers which is closer to what happens in modern computer programming will have many advantages: it may lead to more suitable applied mathematics, and to a better exploitation of computing resources.

Ideological grounds I. One may hold that in the times of computing partial operations are a problematic concept right from the start. A pocket calculator will usually return an error message when asked to compute $1/0$ (a formalisation of that convention is found by working with $1/0 = \bot$), but it won't stop working, wait forever, or need to be thrown away, or to be restarted somehow (although clearing some memory which may be required after performing an unfortunate division comes close to that). Integrating exception handling in datatypes can be a good idea and is a good idea in arithmetic. When motivated by this particular motive, one will prefer having $1/0 = +\infty$ over say $1/0 = 0$.

Ideological grounds II. One may hold that on the basis of philosophical or other non-technical arguments there are deeper reasons to assign a meaning to say $1/0$ or to $0/0$. These deeper reasons are then considered to be quite independent of practical considerations. At least the following positions have support on the basis of ideological considerations:

- $1/0 = +\infty$,
- $1/0 = 0$,
- $0/0 = 0$,
- $0/0 = 1$.

6 Associativity and commutativity preserving alternative views on DbZ

In [6] I have surveyed alternative options for division by zero. The focus of that survey is on AC preserving alternatives with AC standing for associativity and commutativity of addition as well as of multiplication. Here I will provide a summary of that survey while paying more attention to methodological matters. The following options may be distinguished, the list being by no means exhaustive.

6.1 Logic(s) of partial functions

$1/0$ is undefined but there is no objection against the use of expressions like $1/0$ or $2/(3-(1+2))$. This idea underlies various logics of partial functions which have emerged within logic as well as within computer science. This subject is large and I can only give some references which are by no means covering the theme at hand. I mention [22, 40, 29, 30].

The fundamental distinction between the use of a logic of partial functions and the mainstream view is that the expression $1/0$ is considered to feature no fundamental typing problem, which justifies, or even requires its rejection, it merely has no value.

Connections with logic and informatics. Various software specification languages (e.g. VDM and CASL) make use of partial functions and logics of partial functions, a policy which works for division by zero as well.

Comments on logics of partial functions in relation to DbZ. There are different logics of partial functions. So-called Kleene equality identifies $1/0$ with itself (i.e. $1/0 = 1/0$ while in other such logics deriving an equation $t = r$ implies the that both t and r are denoting a value, from which it follows that $x = x$ is not a plausible axiom.

To the best of my knowledge each known logic of partial functions is substantially more complicated than first order logic, or equational logic, i.e. the equational fragment of first order logic.

Now it is tempting to think that first order logic serves as a logical underpinning of school arithmetic but that is not the case. To see this one may consider the following assertion $\Phi \equiv \forall x(x \neq 0 \to x \cdot (1/x) = 1)$ which, I suppose will be agreed to by most school teachers. However for Φ to be true it must (assuming that first order logic is used) hold for all substitutions of x including $x = 0$. Thus $0 \neq 0 \to 0 \cdot (1/0) = 1$ must hold true, which in turn requires $0 \cdot (1/0) = 1$ to have some truth value (either *true* or *false*, both truth values will do), and which requires taking the subexpression $1/0$ seriously somehow.

If one insists on formalising school arithmetic in first order logic it is advisable not to have division as an operation in the signature (otherwise avoiding an expression $1/0$ is impossible) but to work with x/y "as a notation for the unique z such that $y \cdot z = x$ if z exists" and to adopt the rule that assertions involving division must first be translated to equivalent assertions without division before analysing the meaning.

6.2 Four brands of adopting $1/0 = 0$

Adopting $1/0 = 0$ is possible if one takes $\Phi \equiv \forall x(x \neq 0 \to x \cdot (1/x) = 1)$ as an axiom for division and agrees that making division total by taking $x/0 = 0$ is consistent with Φ. Adopting $1/0 = 0$, however, can be done with quite different objectives in mind.

6.2.1 Principled zero-totalisation (division by zero calculus)

Saburoh Saitoh and co-workers claim (since 2014) that adopting $1/0 = 0$ is a profitable idea for mathematics at large and that doing so would constitute a step forward. Principled zero-totalisation is a far reaching assumption and collecting evidence for that position lies outside the scope of this paper. I refer to [34] for work in that direction. Much more work has been done concerning the division by zero calculus. I mention only a few of the available papers: [32, 37, 38, 39]

6.2.2 Pragmatic zero totalisation (Coq, Isabelle, Lean)

The ad hoc position has been taken on board by various informaticians, mainly in the context of theorem proving where avoiding the use of a logic of partial functions is a useful simplification and where adopting $1/0 = 0$ is only a convention which is marginally preferable to adopting say $1/0 = 75$ which would make no difference given that "translating away" division is done anyway.

Examples of proof systems that adopt $1/0 = 0$ are: Coq, Isabelle, Lean. Of course one may claim that these systems merely use a function different from division and have not bothered to choose another name and symbol for it thereby creating some needless confusion about the semantics of division.

6.2.3 Curiosity driven zero totalisation (pseudofields, involutive meadows)

To this category I include Komori [31] and Ono [36]. These papers provide a substantial information on the model theory of pseudofields, i.e. fields equipped with an inverse for which $0^{-1} = 0$. Both papers seem to be the first papers about DbZ containing significant mathematical and logical information. There is no indication that these authors intended to claim any methodological advantages of the use of pseudofields.

Theoretical work on involutive meadows with a logical style belongs to this category, for instance: [13, 9, 8, 11, 20, 21, 12, 5].

6.2.4 Quasi-principled zero-totalisation (involutive meadows)

The theory of (involutive) meadows is a logic style version of the division by zero calculus. This approach is used for the specification of arithmetical datatypes in [17]. The quasi-principled position has been applied in the theory of programming languages for the purpose of axiomatising probabilistic primitives in [35] and for a description of forensic reasoning in [3].

6.3 Quasi-principled $\perp$-enlargement of rationals (common meadows)

In a common meadow (see [14]) one works with the identity $1/0 = \perp$. Here $\perp$ is an peripheral value (number) which is absorptive in the context of numbers.

Connections with computing. If one adopts the idea that an occurrence of $\perp$ in a run constitutes an unsurmountable problem, then the model is quite descriptive for DbZ handling in certain program notations.

- The program notation Algol (as realized by `https://www.tutorialspoint.com/execute_algol_online.php`) conforms the common meadows model by delivering a runtime error on each attempt to divide by 0. Algol 60 deviates from the mainstream position in that expressions like $1/0$ are not rejected at compile time, but whenever a value $\perp$ occurs in a computation that fact causes the run to abort or to raise an exception.
- The same behaviour is observed for Python (`https://www.programiz.com/python-programming/online-compiler/`).
- To the best of my knowledge there is no available theoretical literature which explains in any depth why would be is a significant advantage for a program notation to adopt say $1/0 = +\infty$. The fact that the latter design decision has been repeated so often does not prove beyond reasonable doubt that such an advantage actually exists and it is conceivable that on the long run $1/0 = \Phi$ (representing a non-signalling `NaN` will be generally adopted for program notations. I find it less likely that $1/0 = 0$ will be adopted for program notations, in spite of the fact that it has become the preferred choice for proof checking systems.
- The fundamental strength of common meadows lies in the proximity with the mainstream view, assuming one is willing to understand $\perp$ as "being undefined".

6.4 Quasi-principled totalisation with an unsigned infinity (wheels)

In wheels the idea is that $1/0 = \infty$ where ∞ is an unsigned infinite value. The Riemann sphere serves as an inspiration for the design of Wheels, number systems with a single unsigned infinity coupled with an absorptive value. I label this approach is quasi-principled because the arguments raised in favour of it exceed mere arguments of convenience, and are based on an intrinsic analysis of what division in arithmetic is supposed to be. As references for wheels I mention [41, 23, 24]

There seems to be no manifest application of wheels to informatics. The strength of wheels resides primarily in an adequate rendering of the Riemann sphere in that model.

6.5 Quasi-principled totalisation with signed infinities (transmathematics)

Here $1/0 = \infty$, $(-1)/0 = -\infty$, $1/\infty = 1/(-\infty) = 0$ whereas $0/0$, $0 \cdot (1/0)$, $\infty + (-\infty)$ are identified with an absorptive value (element). This position is elaborated under the name transmathematics in e.g. [2], and [26]. In transmathematics a peripheral entity nullity (written Φ) is used as a name for the absorptive element, (the element which its denoted with $\perp$ in other approaches).

Connection with logic and informatics

- This position is close to DbZ handling in the program notation Pascal (I used the online Pascal environments `https://www.onlinegdb.com/online_Pascal_compiler` and `https://rextester.com/l/Pascal_online_compiler` for experimenting with division). In Pascal ∞ is denoted `+Inf`, $-\infty$ is denoted `-Inf`. `+Inf` equals itself and differs from `-Inf`. Both of these values is printed as output and can be passed as parameter for a procedure. Pascal adopts `Nan + 3 = Nan` etc. as in the setting of Transreals. Operations like `p` := `0.0` $\star$ (`1.0/0.0`) produce error messages. In fact whenever the Transreal value of an expression equals Nullity (i.e. $\perp$ in transmathematics). Pascal will produce a run time error when being asked to evaluate the expression.

  Proposition 6.1. *Ignoring roundoff phenomena including the size of domains* Pascal *(i.e. the mentioned implementation of it) provides a faithful implementation of transreals, under the convention that Nullity (from transreals) is implemented either as an abort with error message or as arriving at the special value* ***Nan*** *.*

  However at closer inspection the handling of DbZ in this Pascal implementation is not easy to follow, with `x, y, z` declared as variables for `single` (and the same for declaration as `double`):

  `x` := `1/0`; `y` := (-1)`/0`; `z` := `x` + `y`; `writeln(z)` aborts, while
  `z` := `1/0` + ((-1)`/0`); `writeln(z)` produces `Nan`.

- The transreal perspective on division by zero is reflected in Java even better than in Pascal, though not completely. At least when using `https://www.jdoodle.com/online-java-compiler/` and if one works with `double` values. An expression `1.0/0.0` inside a program will produce value +`Infinity`, while the expression $(-$`1.0`$)$`/0.0` produces $-$`Infinity`. Terms like `0.0` $\star$ (`1.0/0.0`) produce `NaN`, a value which can be used in computations without any problem. However, in contrast with transreals `NaN` is not equal to itself (i.e. testing `p` == `p` when `p` evaluates to `NaN` yields `false`).

  Proposition 6.2. *Equality of transreals (written as* $\_ =_{tr} \_$*) can be defined in terms of Java equality (in Java written as* $\_ == \_$*) as follows:*

$$x =_{tr} y \iff x == y \vee (\neg x == x \wedge \neg y == y)$$

- In GO we find the same DbZ handling as in Java (using the online tool for GO: `https://play.golang.org`).
- The work on transrationals and transreals (as a part of transmathematics) may be considered to belong to theoretical computer science (datatypes/theory of software engineering).
- In logic similar work is to be expected in connection with constructive analysis but am unaware of approaches to constructive analysis which adopt the same policy on division by zero.

7 Non-associative alternative views on DbZ

Some alternative views of DbZ dispose of associativity of multiplication. This happens when options like $0/0 = 1$ and/or $(1/0) \cdot 0 = 1$ are adopted.

7.1 DbZ in the program notation APL

In the program notation APL as implemented on `replit.com` calculations proceed in unexpected ways.

APL uses its own symbol for division, which I have replaced here by / for consistency of the paper, moreover 1 stands for 1.0 and 0 stands for 0.0; here $t = r$ stands for t evaluates to r, and $\perp$ is represents an abort (i.e. no proper result) with error message (depending on input).

$1/0 = \infty$,
$1 \cdot (1/0) = \perp$,
Apparently 1 is not a left unit for multiplication.
$-(1/0) = -\infty$,
$(-(1/0)) \cdot 1 = \perp$,
Apparently 1 is not a right unit for multiplication.
$-0 = 0$
$1/(-0) = -\infty$,
Apparently APL evaluation equivalence (written say $\_ \equiv_{apl} \_$) is not a congruence: $0 \equiv_{apl} -0$ while $1/0 \not\equiv_{apl} 1/(-0)$.
$0 \cdot (1/0) = 0$,
$(1/0) \cdot 0 = 1$,
Apparently multiplication in APL is not commutative.
$((1/0) \cdot 1) \cdot (-1) = 0$,
$(1/0) \cdot (1 \cdot (-1)) = \infty$,
Apparently multiplication in APL is not associative.

Some other observations on this particular APL implementation:
$(1/0) \cdot (-1) = 0$, $(-1) \cdot (1/0) = \perp$, $1/0 + 1/0 = \infty$, $1/0 \cdot 1/0 = \infty$, $(1/0) \cdot ((-1)/0) = 0$, $(1/0) \cdot (1/(-0)) = 0$, $(1/(-0)) \cdot 0 = \perp$, $(1/0) \cdot (-0) = 1$, $(-0) \cdot (1/0) = 0$, $0 \cdot ((-1)/0) = 0$, $0/0 = \perp$, $1/0 - 1/0 = \perp$.

Originally APL adopts $0/0 = 1$ (see [33]) but that is not the case in the online `replit.com` environment. Still a residue of that identity is visible in $(1/0)\cdot 0 = 1$. I have not been able to discover the rationale behind the behaviour of this APL implementation regarding DbZ.

7.2 Principled $0/0 = 1$ adoptation.

In [42, 43] grounds are formulated for adopting $0/0 = 1$, I am unconvinced by these arguments, but undeniably having grounds for accepting or rejecyting that thesis are sound.

7.3 Curiosity driven $0/0 = 1$ adoptation

I have not found literature on curiosity driven work based on the assumption that $0/0 = 1$. Nevertheless I include this option because it potentially constitutes a path towards further novel work on DbZ. A price to be paid is that multiplication will not be associative (see [4] for more on that assumption), but doing so may be worth the effort.

8 Concluding remarks

The concluding remars are organised as conclusion regarding the three areas where DbZ may be of relevance: mathematics, logic and informatics (or more specifically computing).

8.1 DbZ in mathematics.

However, in view of the large number of alternative positions it is hard to imagine that either of these will gain world-wide prominence in the near future. I expect that the future of DbZ will allow room for different views on the matter. When it comes to spotting an alternative perspective on DbZ which is useful within mathematics I see 3 candidates: (i) division by zero calculus, (ii) transmathematics, and (iii) logic(s) of partial functions. A discussion of the potential of use of these approaches lies outside the scope of this paper, the focus of which is on logic, and on computing. However, there is an area in between of mathematics, logic and computing to which I want to draw special attention: the principles of school arithmetic.

8.2 DbZ in logic.

Logical work on DbZ has thus far been carried out in three directions: (i) logics of partial functions, (ii) involutive meadows, and (iii) common meadows.

Nevertheless, it is fair to say that as far as mathematical logicians are concerned and for disparate reasons DbZ handling does not pose a problem for them which is worth of much attention. This lack of interest in DbZ related

matters may be connected with a focus on first order logics (and stronger logics) rather than on conditional equational logic.

8.3 DbZ in computing

Regarding DbZ in computing I will formulate 7 more specific conclusions and two open problems.

1. $1/0$ **is denoting.** In as far as one considers DbZ an empirical matter it matters to look at today's plurality of programming languages: it is the rule rather than the exception that a computation involving an evaluation of (say) $1/0$ will carry on without aborting or raising an exception, and it is a rule rather than an exception that $1/0$ evaluates to a positive value understood as a positive infinite value (i.e. larger than all ordinary values).

2. **Peripheral numbers.** Datatypes for number systems able to model the current practice of computer programming are bound to possess one or more what I call peripheral numbers. Here are 5 relatively well-known peripheral numbers, more of these can be imagined or extracted from computing practice:

   ∞ (unsigned infinite value),

   $+\infty$ (positive infinite value),

   $-\infty$ (negative infinite value),

   $\perp$ (signalling NaN), and,

   Φ (silent/nonsignalling NaN).

   Here Φ is the constant called Nullity in the transrationals/transreals of [2]. The peripheral numbers $\infty, +\infty, -\infty$ are formal infinities, to be distinguished from the actual infinity of the size of an non-finite set and the potential infinity of the limit of the sequence of natural numbers.

3. **(Re)naming matters.** When writing [18] no attention was paid to the difference in status that one may attribute to $\perp$ and ϕ. In hindsight naming matters. In that paper an abstract datatype $\mathbf{Q}_{tr}$ is specified (that is an isomorphism class of datatypes say containing datatype say Q_{tr}).

   As a consequence it is only after renaming $\perp$ into Φ that the abstract datatype $\mathbf{Q}_{tr}$ captures the essence of transrationals completely. Following module algebra notation (from [10]) the renaming function can be denoted $\rho_{(\perp,\Phi)}$ (a permutation of the constants names $\perp$ and Φ) and we may write:

   $$Q_{transrat} = \rho_{(\perp,\Phi)}(Q_{tr}) \text{ and } \mathbf{Q}_{transrat} = \rho_{(\perp,\Phi)}(\mathbf{Q}_{tr})$$

   for the appropriately renamed datatype and abstract datatype respectively.

4. **Prominence of transrationals.** The only datatype which has (until now) been designed, analysed, and published with the intention to understand how DbZ works in program notations of later date than Algol 60 is the datatype of transreals (or its subsystem of transrationals). For transreals I refer to [2], and [26] and references cited in those papers. However the following additional remarks are in order:

   (i) I am unaware of a program notation which handles DbZ precisely as it works in transreals (though Java comes quite close). However, Pop-11 (see e.g. [1]) can make use of a library so that its floating point system then works just as prescribed by transreal arithmetic (at least regarding matters of DbZ).

   (ii) Finding how and where a program notation deviates (regarding its DbZ handling) from transreals provides a useful perspective on how DbZ has been implemented in that specific program notation. In other words the datatype of transreals captures the complexities involved in the practice of DbZ quite well.

   (iii) I have not yet found a program notation which deviates from the datatype of transreals and which at the same give rise to the design of a (reasonably attractive) modified datatype for numbers with DbZ which is better suitable for an explanation of how DbZ works in that specific program notation.

   (iv) The IEEE 754 floating point standard shows manifest similarities with transreals (regarding DbZ) but is at the same considerably more complicated than design of transreals. More specifically IEEE 754 seems to assign status and relevance to negative `NaN` values, an idea which lies entirely outside the scope of transrationals. A datatype which may underly IEEE 754 has not yet been designed and may not exist. An attempt to model negative `NaN`'s was made in [4] but the resulting datatype is not very illuminating.

   (v) I am unaware of any suggestions in the current literature on how to extend, modify, or improve the design of the datatype of transrationals (transreals) in such a manner that a better fit with "observations in the field" is obtained.

5. **Combining $\bot$ and Φ.** With $\mathsf{Enl}_{\bot}(A)$ I denote the enlargement of datatype A with an absorptive element (and constant) $\bot$. If the signature of A contains already a constant named $\bot$, A is left unchanged by the transformation $\mathsf{Enl}_{\bot}(A)$. Now the datatype of transrationsals as mentioned in item 3 above can be enlarged in this way to obtain a datatype $Q_{transrat,\bot}$:

$$Q_{transrat,\bot} = \mathsf{Enl}_{\bot}(Q_{transrat})$$

   Subsequently one may develop enrichments of $Q_{transrat,\bot}$ in order to develop datatypes which specify how DbZ works in various program notations. For any datatype the signature of which contains 0 and 1 one may

introduce a function $\_ == \_ :\to z$ such that $x == x :\to z$ equals z for all x (i.e. $x == x :\to z = z$) and $x == y :\to y$ equals 0 for pairs x, y so that $x \neq y$ (i.e. $x \neq y \to x == y :\to z = 0$). I write $\mathsf{enr}_{== :\to}(A)$ for the enrichment of datatype A with the function $\_ == \_ :\to \_$ (which is included in the signature at the same time. If a function $\_ == \_ :\to \_$ happens to be present in the signature of A already then A is left unaffected.) The datatype $Q_{transrat,\perp,== :\to}$ given by

$$Q_{transrat,\perp,== :\to} = \mathsf{Enr}_{== :\to}(Q_{transrat,\perp})$$

provides the expressive power to describe the plurality of patterns of handling DbZ as observed when experimenting with various program notations/environments.

For instance consider a program notation PN that features an implementation of the division function $\_/\_$ which aborts as soon as division by 0 occurs (ALGOL 60 works like that). This state of affairs can be modeled by writing:

$$x /_{\mathsf{PN}} y = x/y + (y == 0) :\to \perp + (y == \Phi) :\to \perp + (y == \perp) :\to \perp$$

6. **Application area for datatype theory.** DbZ (as it occurs in the practice of computer programming) is an attractive area for the use of datatypes and abstract datatypes (see e.g. [28] for an introduction). With the design of an algebraic specification of the abstract datatype of transrational numbers a first step in that direction has been made in [18] following the approach of the considerably simpler algebraic specification of rationals in [17], with the specification of the abstract datatype of wheels of rationals (as presented in [19]) as an intermediate step). The complexity of the world of expressions in transrational arithmetic is highlighted in [6], whereas the potential of transreal analysis is convincingly illustrated in [27].

7. **Definition of peripheral numbers.** For the discussion in this paper $0 \cdot x \neq x$ serves as an appropriate definition of a number being peripheral. That definition must be revised, however, upon the introduction of an inifinitesimal constant ι which satisfies $0 \cdot \iota = 0$ while it may also satisfy $\iota + (-\iota) = \perp$ (or $\iota + (-\iota) = \Phi$). A "better" definition of x being peripheral is $x + (-x) \neq 0$, as in Definition 4.1.

8. **Open problems on term rewriting in the presence of $\perp$.** The abstract datatype

$$\mathsf{ADT}(\mathsf{Enl}_{\perp}(Q(\_/\_)))$$

is computable and it allows an initial algebra specification (hinted at in [15]). With [16] it follows that with the help of auxiliary functions an initial algebra specification can be given which constitute a confluent and terminating term rewriting system.

Problem 8.1. *Does* $\mathsf{ADT}(\mathsf{Enl}_{\perp}(Q(\_/\_)))$ *a possess an initial algebra specification which viewed as a term rewriting system is confluent and terminating?*

An equally fundamental and open question is the following:

Problem 8.2. *Is the equational theory of* $\mathsf{ADT}(\mathsf{Enl}_{\perp}(Q(\_/\_)))$ *decidable?*

8.4 DbZ in teaching arithmetic

The following conclusions may be of relevance for the educational science of teaching arithmetic.

1. **Mainstream in doubt.** The certainty shown by many proponents of the mainstream perspective on DbZ, namely that such a phenomenon may and even must be rejected by all means, including ridiculing those who look for alternative options to DbZ, is an illusion. The topic is complicated and with computing increasingly finding a foothold in mathematics at all levels it is just a matter of time that the mainstream position will be given up in favour of a more flexible perspective on DbZ.

2. **For primary education:** $1/0$ **is a fracterm without meaning.** In primary education I propose that it can be said that $1/0$ is a fracterm which will not be given a meaning, at least not in the near future. In various branches of logic and computing, however, so may be said in addition, it is customary to assign a meaning to $1/0$. (The notion of a fracterm is introduced in [15] and is developed further in [7]. The latter paper uses a common meadow of rationals as its semantic basis)

   The situation is comparable with $\sqrt{-1}$ which can be imagined by students who are familiar with say $\sqrt{4} = 2$ long before they are told what $\sqrt{-1}$ might mean in an extension of the number system they are looking at as students when becoming familiar with the $\sqrt{\ }$ sign.

   When asked what sort of value $1/0$ can take it may be said that:

   (i) Mathematicians and logicians who use computers for doing mathematics (in particular proof generation and proof checking) frequently opt for adopting $1/0 = 0$, just for convenience as it turns out. Significant communities are making use of $1/0 = 0$ for these reasons.

   (ii) For computing at large and mathematical modelling of dynamic systems (so-called number crunching) working with $1/0 = +\mathtt{Inf}$ is the most popular choice (which comes as no surprise in view of the fact that IEEE 754 floating point standard prescribes that way of going about DbZ).

3. **DbZ in the foundations of elementary arithmetic.** I hold that elementary arithmetic (mathematics) as it is taught in primary education is not identical to merely a subset of professional arithmetic (mathematics). More specifically I hold that there is more room and need for a syntactic

perspective at the elementary level. Symbols and expressions start their life in the mind of a human agent long before a mature perspective on semantics has been acquired. These initial phases allow being studied from the perspective from the subject matter at hand. What may be considered an immature view on a piece of professional mathematics may potentially be framed as a mature view on a different though similar content.

Given its massive and ubiquitous presence world-wide, and in the light of the formidable investments in time and money made in arithmetical education it is reasonable to pursue the development of proper foundations of school arithmetic as a subject matter, that is a focus on what is taught quite independently of how it is taught, on why it is taught or on when it is taught.

DbZ, or rather policies for avoiding DbZ are among the core themes of elementary arithmetic. I expect that the common meadows ([14]) of rationals will provide a suitable basis for further development of foundations of school arithmetic. An example of such work is [7] in which the notion of a fraction is scrutinized.

4. **Common meadows of rationals: tools for explaining the foundations of school mathematics.** The educational relevance of DbZ is highlighted in detail in [25]. Common meadows of rationals (i.e. working with $1/0 = \bot$) have a role to play in the formalization of the mainstream way of dealing with DbZ. The idea is that the rules of engagement concerning DbZ as embodied in the mainstream approach can be profitably explained by way of a translation of a mathematical story about $Q(\_/\_)$ (a datatype of rational numbers with a partial division operator), to $\mathsf{Enl}_\bot(Q(\_/\_))$, the $\bot$ enlargement of $Q(\_/\_)$. In principle such a translation can be given just as well into a setting where $1/0 = 0$ but working with $\mathsf{Enl}_\bot(Q(\_/\_))$ is far closer to the intuition of partiality of division. The key advantage of the translation to $\mathsf{Enl}_\bot(Q(\_/\_))$ is that subsequently first order logic (FOL) and its equational fragment, equational logic, can be used without hesitation. Some details require attention, however.

   Given an expression t for $\Sigma(Q(\_/\_))$ (i.e. the signature of unital rings augmented with a name for division), $\mathsf{var}(t)$ is the collection of variables occurring in t and $\Sigma_{\mathsf{var}(t)}$ is the sum of the variables in $\mathsf{var}(t)$ if that set is nonempty and 0 if it is empty. (The use of Σ for sum and for signature is wholly unrelated). Now consider an equation $t = r$ then we may have in mind a notion of $t = r$ being valid from the mainstream perspective:

$$Q(\_/\_) \models_{mainstream} t = r$$

   Then $\models_{mainstream}$ can be explained by the following equivalence:
   $Q(\_/\_) \models_{mainstream} t = r$ if and only if the following holds:

$$\mathsf{Enl}_\bot(Q(\_/\_)) \models 0{\cdot}t = 0{\cdot}\Sigma_{\mathsf{var}(t)} \wedge 0{\cdot}r = 0{\cdot}\Sigma_{\mathsf{var}(r)} \wedge t + 0{\cdot}\Sigma_{\mathsf{var}(r)} = r + 0{\cdot}\Sigma_{\mathsf{var}(t)}$$

Now it follows for instance that: $Q(\_/\_) \models_{mainstream} 0 \cdot x = 0$. Formalisation of (DbZ) mainstream style proofs require dealing with assertions of the form:

$$Q(\_/\_) \models_{mainstream} \phi \to t = r$$

where ϕ expresses the combination of assumptions which have been made (both in the statement of the fact to be shown, as during the proof) about the variables occurring in t and in r. Expanding on the details of (explaining the meaning of) conditional reasoning about partial functions lies outside the scope of this survey.

It should be noticed, however, that making use of the assumption $1/0 = \perp$ for explaining the details of the mainstream view by itself caries little weight when arguing that $0 \cdot x = x$ is defeasible. Indeed mainstream reasoning with partial division can be given foundations by other means for instance along the lines of [40] which is not based on the totalisation of functions.

5. **Determination of $1/0$ is a matter of design, not of science.** Assigning a value to $1/0$ is part of the design of arithmetical language, a design which may be tailor made for specific conditions and objectives. Different objectives call for different arithmetical languages thereby leading to differences in dealing with DbZ.

   Under the assumption (which I subscribe to myself) that determination of $1/0$ by choosing from a menu of options (including the mainstream view of DbZ) is a matter of design, there is no merit in asking for scientific (mathematical, logical, philosophical) consensus on the value (if any) of $1/0$, and there is no substance in claiming the existence of such consensus.

   Adopting axioms is a matter of design just as the determination of certain values is. Adopting $x \geq 0$ amounts to the exclusion of negative numbers, Adopting $x^2 \geq 0$ amounts to the exclusion of (non-real) complex numbers; adopting $x \cdot y = y \cdot x$ excludes quaternions; adopting the Archimedian principle excludes non-standard numbers. In the same vein adopting the axiom $0 \cdot x = 0$ amounts to excluding the peripheral numbers $\infty, +\infty, -\infty, \perp, \Phi$.

Acknowledgement. James Anderson (Reading, UK) has made a number quite helpful and detailed comments and suggestions concerning an earlier draft of this paper. Kees Middelburg (Voorschoten, NL) and Alban Ponse (Amsterdam NL) gave useful feedback as well.

References

[1] James A.D.W. Anderson (ed.). Pop–11 Comes of Age; the advancement of an AI Programming Language. Ellis Horwood, (1989).

[2] J.A. Anderson, N. Völker, and A. A. Adams. Perspecx Machine VIII, axioms of transreal arithmetic. Vision Geometry XV, eds. J. Latecki, D. M. Mount and A. Y. Wu, 649902, (2007).

[3] J.A. Bergstra. Adams conditioning and Likelihood Ratio Transfer Mediated Inference. *Scientific Annals of Computer Science*, 29 (1), 1–58, (2019).

[4] J.A. Bergstra. Division by zero, a survey of options. *Transmathematica*, ISSN 2632-9212, (published 2019-06-25), `https://doi.org/10.36285/tm.v0i0.17`, (2019).

[5] J.A. Bergstra. Dual number meadows. *Transmathematica*, ISSN 2632-9212 (published 06-25-2019), `https://doi.org/10.36285/tm.v0i0.11`, (2019).

[6] J.A. Bergstra. Fractions in transrational arithmetic. *Transmathematica*, ISSN 2632-9212 (published 01-27-2020), `https://doi.org/10.36285/tm.19`(2020).

[7] J.A. Bergstra. Arithmetical datatypes, fracterms, and the fraction definition problem. *Transmathematica*, ISSN 2632-9212, (published 2020-04-30), `https://doi.org/10.36285/tm.33`, (2020).

[8] J.A. Bergstra, I. Bethke. Subvarieties of the variety of meadows. *Scientific Annals of Computer Science.* vol 27 (1) 1–18, (2017).

[9] J. A. Bergstra, I. Bethke, and A. Ponse. Equations for formally real meadows. *Journal of Applied Logic*, 13(2): 1–23, (2015).

[10] J.A. Bergstra, J. Heering, and P. Klint. Module algebra. *Journal of the ACM*, 37 (2): 335–372, (1990).

[11] J.A. Bergstra, Y. Hirshfeld, and J.V. Tucker. Meadows and the equational specification of division. *Theoretical Computer Science*, 410 (12), 1261–1271, (2009).

[12] J.A. Bergstra and C.A. Middelburg. Transformation of fractions into simple fractions in divisive meadows. *Journal of Applied Logic*, 16: 92–110, also `https://arxiv.org/abs/1510.06233`, (2015).

[13] J.A. Bergstra and J.V. Tucker. Division safe calculation in totalized fields. *Theory of Computing Systems*, 43, 410–424 (2008).

[14] J.A. Bergstra and A. Ponse. Division by zero in common meadows. *In R. de Nicola and R. Hennicker (editors), Software, Services, and Systems (Wirsing Festschrift),* LNCS 8950, pages 46-61, Springer, 2015. Also available at `arXiv:1406.6878v4 [math.RA](2021)` an improved version the original paper, (2015).

[15] J.A. Bergstra and A. Ponse. Fracpairs and fractions over a reduced commutative ring. *Indigationes Mathematicae* 27, 727-748, (2016). `http://dx.doi.org/10.1016/j.indag.2016.01.007`. (Also `https://arxiv.org/abs/1411.4410`).

[16] J.A. Bergstra and J.V. Tucker. Equational specifications, complete term rewriting systems, and computable and semicomputable algebras. *Journal of the ACM*, Vol. 42 (6), 1194-1230 (1995).

[17] J.A. Bergstra and J.V. Tucker. The rational numbers as an abstract data type. *Journal of the ACM*, 54 (2), Article 7, (2007).

[18] J.A. Bergstra and J. V. Tucker. The transrational numbers as an abstract datatype. *Transmathematica*, ISSN 2632-9212, (published 2020-12-16), `https://doi.org/10.36285/tm.47`, (2020).

[19] J.A. Bergstra and J. V. Tucker. The wheel of rational numbers as an abstract datatype. Proceedings of the Workshop on Algebraic Development Techniques, Swansea University, 2020, to appear (2021).

[20] I. Bethke and P.H. Rodenburg. The initial meadows. *Journal of Symbolic Logic*. 75 (3), 888-895, (2010).

[21] I. Bethke, P.H. Rodenburg and A. Sevenster. The structure of finite meadows. *Journal of Logical and Algebraic Methods in Programming,* 84 (2), 276-282, (2015).

[22] Manfred Broy and Martin Wirsing. Partial abstract types. *Acta Informatica,* 18, 47–64, (1982).

[23] J. Carlström. Wheels–on division by zero. *Math. Structures in Computer Science*, 14 (1) pp. 143–184, (2004).

[24] J. Carlström. Partiality and choice, foundational contributions. *PhD. Thesis, Stockholm University*, `http://www.diva-portal.org/smash/get/diva2:194366/FULLTEXT01.pdf`, (2005).

[25] Sandra Crespo and Cynthia Nicol. Challenging preservice teacher's mathematical understanding: the case of division by zero. *School, Science and Mathematics*, 106 (2), 88–97, (2006).

[26] T.S. dos Reis, W. Gomide, and J.A.D.W. Anderson. Construction of the transreal numbers and algebraic transfields. *IAENG International Journal of Applied Mathematics*, 46 (1), 11–23, (2016). (`http://www.iaeng.org/IJAM/issues_v46/issue_1/IJAM_46_1_03.pdf`)

[27] Tiago Soares dos Reis. Transreal integral. *Transmathematica*, ISSN 2632-9212, (published 2019-06-25), `https://doi.org/10.36285/tm.v0i0.13`, (2019).

[28] H. D. Ehrich, M. Wolf, and J. Loeckx. Specification of Abstract Data Types. *Vieweg + Teubner*, ISBN-10:3519021153, (1997).

[29] Antonio Gavilanes-Franco and Francisa Lucio-Carasco. A first order logic for partial functions. *Theoretical Computer Science*, 74: 37–69, (1990).

[30] C. B. Jones and C. A. Middelburg. A typed logic of partial functions, reconstructed classically. *Acta Informatica*, 31, 399–430, (1994).

[31] Y. Komori. Free algebras over all fields and pseudo-fields. Report 10, pp. 9-15, Faculty of Science, Shizuoka University, (1975).

[32] Tsutomu Matsuura and Saburou Saitoh. Matrices and division by zero $z/0 = 0$. *Advances in Linear Algebra & Matrix Theory*, 6 (2) Article ID 67301, (2016).

[33] E. E. McDonnel, Zero divided by zero. APL 1976 Proceedings, 295–296 (`https://doi.org/10.1145/800114.803689`), (1976).

[34] H. Michiwaki, S. Saitoh, and N. Yamada. Reality of the division by zero $z/0 = 0$. *International Journal of Applied Physics and mathematics*, doi: 10.17706/ijapm.2016.6.1.1-8 (2016).

[35] C. A. Middelburg. Probabilistic process algebra and strategic interleaving. *Scientific Annals of Computer Science*, 30 (2), 205-243, (2020).

[36] H. Ono. Equational theories and universal theories of fields. *Journal of the Mathematical Society of Japan*, 35(2), 289-306, (1983).

[37] Hiroshi Okumura, Saburou Saitoh and Tsutomu Matsuura. Relations of zero and ∞. *Journal of Technology and Social Science* 1 (1), (2017).

[38] Hiroshi Okumura. Is it really impossible to divide by zero? *Biostatistics and Biometrics Open Acc. J.* 7 (1) 555703. DOI: 10.19080/BBOJ.2018.07.555703, (2018)

[39] Hiroshi Okumura and Saburou Saitoh. Applications of the division by zero calculus to Wasan geometry. *Global Journal of Advanced Research on Classical and Modern Geometries*, 7 (2), 44–49, (2018).

[40] Anthony Robinson. Equational logic of partial functions under Kleene equality: a complete and an incomplete set of rules. *Journal of Symbolic Logic*, 54 (2): 354–362, (1989).

[41] Anton Setzer. Wheels (draft). `http://www.cs.swan.ac.uk/csetzer/articles/wheel.pdf`, (1997).

[42] Okoh Ufuoma. On the operation of division by zero in Bhaskara's frame-work: criticisms, modifications, and justifications. Asian Research Journal of Mathematics, 6 (2), 1–20, Article no. AJROM.36240 (2017). DOI:10.9734/ARJOM/2017/36240.20m (2017).

[43] Okoh Ufuoma. Exact arithmetic of zero based on the conventional division by zero $0/0 = 1$. Asian Research Journal of Mathematics, 13 (1), 1–40, `https://doi.org/10.9734/arjom/2019/v13i130099`, 13 (1), (2019).