Partial arithmetical data types of rational numbers and their equational specification

Bergstra, J.A.; Tucker, J.V.

DOI
10.1016/j.jlamp.2022.100797

Publication date
2022

Document Version
Final published version

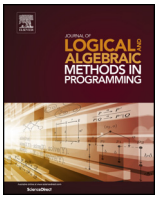
Published in
Journal of Logical and Algebraic Methods in Programming

Journal of Logical and Algebraic Methods in Programming 128 (2022) 100797

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp



Partial arithmetical data types of rational numbers and their equational specification

Jan A. Bergstra [a], John V. Tucker [b,*]

[a] Informatics Institute, University of Amsterdam, Science Park 904, 1098 XH, Amsterdam, The Netherlands
[b] Department of Computer Science, Swansea University, Bay Campus, Fabian Way, Swansea, SA1 8EN, United Kingdom

ARTICLE INFO

Article history:
Received 15 August 2021
Received in revised form 24 July 2022
Accepted 26 July 2022
Available online 1 August 2022

Keywords:
Partial data types
Rational numbers with division
Common meadows
Involutive meadows
Fracterm calculus

ABSTRACT

Upon adding division to the operations of a field we obtain a meadow. It is conventional to view division in a field as a partial function, which complicates considerably its algebra and logic. But partiality is one out of a plurality of possible design decisions regarding division. Upon adding a partial division function $\div$ to a field Q of rational numbers we obtain a partial meadow $Q(\div)$ of rational numbers that qualifies as a data type. Partial data types bring problems for specifying and programming that have led to complicated algebraic and logical theories – unlike total data types. We discuss four different ways of providing an algebraic specification of this important arithmetical *partial* data type $Q(\div)$ via the algebraic specification of a closely related *total* data type. We argue that the specification method that uses a common meadow of rational numbers as the total algebra is the most attractive and useful among these four options. We then analyse the problem of equality between expressions in partial data types by examining seven notions of equality that arise from our methods alone. Finally, based on the laws of common meadows, we present an equational calculus for working with fracterms that is of general interest outside programming theory.

1. Introduction

On applying a function f to an argument x a computer computation is expected to return an answer $f(x)$ quickly. In particular, it is not expected to fail to return an answer and so force the computation to wait indefinitely and ultimately spoil the computation. Said differently, since computer computations cannot wait, the function f needs to be total and not partial. Computers need to have a total semantics for f in all situations, which leads to having special data to flag undefined values.

From the user's point of view, the absence of a value of f has significant consequences for the theory to which f belongs. In all cases, the algebraic and logical reasoning becomes much more complicated because of the effects of f being undefined in expressions and programs. Problems arise with the possible meaning of equality, when two algebraic expressions are declared equal. In programming these problems are everywhere that partiality is encountered and lead to various semantical interpretations and theories. This effect is seen most easily in the algebraic specification of data types with partial operations, which is much more complicated than that for data types with total operations.

* Corresponding author.
E-mail addresses: j.a.bergstra@uva.nl, janaldertb@gmail.com (J.A. Bergstra), j.v.tucker@swansea.ac.uk (J.V. Tucker).

https://doi.org/10.1016/j.jlamp.2022.100797

In this paper, we examine the algebraic specification of division in fields and the specification of arguably the most important partial data type, that of the rational numbers with division $Q(\div)$. For an arithmetical data type with division, the problem is to create a specification that

(i) controls the values of $x/0$, and

(ii) serves as an axiomatisation of all the arithmetical operators that is recognisable, informative and useful.

In the case of arithmetical data types, condition (ii) is rather important since the properties of most of the operators have been familiar since childhood.

We will turn to the specification theory of total data types. From the point of view of data type specification, the general methods we explore have this form:

1. Given a partial data type A with signature Σ, devise a closely related total data type B of signature Σ'.
2. Create an equational specification (Σ', E) of B, if possible.
3. Transform B to recover the partial data type A.

Our construction of B for A will be straightforward algebraically; however, difficulties may lie in the existence and user appeal of the specifications of B. Our standard technique for making algebraic specifications for B is constructing an initial algebra $T(\Sigma', E)$ for the specification (Σ', E) via congruences on terms. We present four methods to specify A with different choices for B.

For arithmetic structures, and specifically a partial data type $Q(\div)$ of rational numbers, we have in mind a number of candidates for B, all of which have interesting and non-trivial equational specifications. Practical computation has designed a number of total arithmetical data types in which $1/0$ is defined – by using flags such as error, ∞, NaN – the last standing for 'not a number'. These arithmetical data types have been studied over the years with languages and standards in mind, but only recently systematically using the methods of algebraic specification theory. The candidates for B are all examples of total data types for numerical computation that have been equationally specified and include:

Involutive meadows, where an element of A's domain is used for totalisation, in particular $1/0 = 0$, [21].

Common meadows, where a new external element $\bot$ that is 'absorbtive' is used for totalisation $1/0 = \bot$, [17]; this data type models errors in common calculators.

Wheels, where an external ∞ is used, together with an additional external error element $\bot$ to help control the effects of infinity, and $1/0 = \infty$, [23]; this data type models exact numerical computation.

Transrationals, where besides the error element $\bot$ two external signed infinities are added, one positive and one negative, so that division is totalised by setting $1/0 = \infty$ and $-1/0 = -\infty$, [22]; this data type models a semantics of floating point computation.

We will discuss these total data types in due course. Through our results, it will become clear that the method based on choosing a common meadow for B is best suited for the task. A survey of options for division by zero is [4,8].

At the technical heart of the theory of algebraic specifications of data types is the notion of equality $=$ and the rules for working with equations, i.e., equational logics. But when operations are partial, equality can have a number of interpretations and, hence, the meaning of equations and the rules they obey can vary and become very complicated [3,45]. Indeed, it is this complexity that prompts the problem we have formulated and studied here. Our specification methods generate seven notions of equality. In the matter of equality, our analysis suggests that

Partiality is not a semantic problem because of the absence of values for an operation. It is a problem because of the effect it has on the meaning of equality between syntactic expressions.

Finally, picking up the requirement that a specification should be recognisable, informative and useful, we consider equational calculi for calculating with fractions that are of interest outside programming, e.g., relevant to teaching that addresses division by zero. Fractions, although everywhere, do not have clear rigorous mathematical definitions. So, we define syntactic terms called *fracterms*, and we propose an equational calculus of fracterms, based on the rules for common meadows. A property of this fracterm calculus for common meadows is simplifying a general fracterm to single fracterm with only one division, a process called *flattening*. We show that flattening is not possible without the addition of an element to the rational numbers. Thus, it is not possible for a calculus for fractions based on the option $1/0 = 0$, discussed by Suppes [47,1] and first analysed by Ono [42].

To sum up, the main contributions of this paper are:

1. Methods for specifying partial data types using total data types.
2. Evaluation of the techniques based on technical criteria, especially algebraic simplicity and computability, and their useablity.
3. An equational specification of the rationals with division.
4. A survey of equality for partial data types.
5. A fracterm calculus for calculating with rational numbers.
6. A comparison of the fracterm calculus for common meadows with a fracterm calculus based on $1/0 = 0$.

The structure of the paper is this. Section 2 sets out some basic notions and notations for the theory of algebraic specifications. Section 3 describes the four methods. Section 4 looks at the implications of partiality through the lens of

computability. Section 5 discusses the equational specification of our chosen candidates for total algebras. Section 6 is about the partial equalities induced from the total algebras. Section 7 uses the best method to propose a fracterm calculus. Section 8 recalls some of the older techniques of partial specifications.

2. Preliminaries

2.1. Basics

We assume the reader is familiar with the theory of algebraic specifications for data types [31,32,50,37].

Definition 2.1. An algebra A is Σ-*minimal* if it is generated by the constants and operations named in Σ. Alternately, A is Σ-minimal if each element of the domain A is the interpretation in A of a closed or ground Σ term.

Definition 2.2. A *partial data type* is an algebra A of signature Σ that is Σ-minimal. A *partial abstract data type* is the isomorphism class $\mathbf{A}$ of a partial data type A.

The condition of minimality is important. It means that every element of the data type can be constructed by applying the operations of the data type to its named constants.

As we are focussed on arithmetic, for simplicity when speaking of general principles, we only consider single-sorted data types with signature Σ; in the case of many sorts similar general observations can be made.

In this paper we examine the application of algebraic specification methods to partial abstract data types. The topic is not new – see Section 8.2. However, the theory of partiality is far from simple and stable.

Let $T(\Sigma)$ be the set of all closed terms made from the operations and constants of Σ.

For simplicity, we will work with data types rather than abstract data types. To specify a data type, we will use equations with the standard initial algebra semantics for total data types. To specify a total data type B by an equational specification (Σ, E) we make an initial algebra using the concrete construction:

$$T(\Sigma, E) = T(\Sigma)/\equiv_E \cong B$$

where

$$t \equiv_E t' \iff E \vdash t = t'.$$

The initial algebra semantics of (Σ, E) is the isomorphism class $\mathbf{I}(\Sigma, E)$ of $T(\Sigma, E)$.

2.2. Transforming data types

We will need to make some transformations of algebras. Suppose $\Sigma \subset \Sigma'$. The Σ-reduct A of the Σ' algebra B is the algebra obtained from B by including only the constants and operations of B named in Σ. Using the standard operations and notations of Module Algebra [13], the reduct is denoted $A = \Sigma \square B$.

The operation $\rho_{(\alpha,\beta)} : \Sigma \to \Sigma$ permutes the names α and β of constants or operations in a signature Σ.

Let $T(\Sigma, X)$ be the set of all terms made from the operations and constants of Σ with variables from X.

An explicit definition of a function has the form

$$F(x_1, \ldots, x_n) = t(x_1, \ldots, x_n),$$

where $t \in T(\Sigma, x_1, \ldots, x_n)$. Note that adding such a definition makes the function F equationally definable over Σ. We will need to use explicit definitions involving conditional operators. We use the 4-place algebraic operator

$$f(u, v, x, y) = x \lhd u = v \rhd y$$

which, in a familiar programming style, means

$$f(u, v, x, y) = \text{if } u = v \text{ then } x \text{ else } y.$$

A data type A can be changed by adding or removing constants and operations *without* changing its domain; these internal changes we call *expansions* or *reductions*, respectively. A data type A can also be changed by adding or removing constants and operations that require adding or removing elements of its domain; these external changes we call *extensions* or *restrictions*, respectively. Extensions followed by expansions we call *enlargements*. The latter are important in what follows, where we will use the following transformations to increase and reduce domains.

First, let us make a total algebra from a partial algebra. At first sight, the easiest way is to use an element from the partial algebra to totalise the partial operations. Let A be a partial minimal algebra of signature Σ. Let $t \in T(\Sigma)$ be a closed term such that t has a value in A. Then we define the Σ algebra

$$B = \mathsf{Tot}_t(A)$$

to be the total algebra obtained by using the value of t in A to make the partial operations of A total by returning the value t. Typically, we can use an existing constant from Σ.

Alternately, we can add a new element to make the partial operations total; however, we must define all the operations on any such new element. An easy way to do this is to choose a new element $\perp \notin A$ that is *absorbtive*, i.e., if $\perp$ is presented as an argument to an operation of A then its value is $\perp$.

Let A be a partial minimal algebra of signature Σ. Let $\perp$ be an element new to A. Then we define the

$$B = \mathsf{Enl}_\perp(A)$$

to be the total algebra obtained by using the value of $\perp$ in A to make partial operations of A total by returning the value $\perp$ and ensuring that $\perp$ is absorbtive.

Just as we make a total algebra from a partial algebra, we can create a partial algebra from a total algebra. Let B be a total minimal algebra of signature Σ'. Let $t \in T(\Sigma')$ be a closed term. Then we define

$$A = \mathsf{Pdt}_t(B)$$

to be the partial algebra obtained by removing the value of t from B making operations of B partial whenever they return the value of t. We define multiple removals step by step, e.g.,

$$A = \mathsf{Pdt}_{t_1,t_2}(B) = \mathsf{Pdt}_{t_1}(\mathsf{Pdt}_{t_2}(B)).$$

Typically, we will use these operations with the constants added to make total functions.

The operators are connected, they are particularly useful in this form (cf. [24]):

Proposition 2.1. *Let B be a total algebra with $\perp$ as an absorptive element, and such that B has at least two elements. Then*

$$\mathsf{Enl}_\perp(\mathsf{Pdt}_\perp(B)) = B.$$

Proposition 2.2. *Let A be any algebra and suppose $\perp \notin A$. Then*

$$\mathsf{Pdt}_\perp(\mathsf{Enl}_\perp(A)) = A.$$

For a closed term t and a total data type B the partial algebra $A = \mathsf{Pdt}_t(B)$ may not be minimal.

Example 2.1. *Let $N = (\{0, 1, 2, \ldots\}, S(-), 0)$ be an algebra of natural numbers. Then $\mathsf{Pdt}_{S(S(0))}(N)$ has an infinite domain, but only two of its elements are the interpretation of a closed term.*

2.3. Arithmetic data types

The basic algebra of arithmetic is the algebra of rings and fields. These axiomatise the operations of addition $x + y$, its additive inverse $-x$, and multiplication $x.y$. Let G be a field with signature Σ.

Although each non-zero element $x \in G$ of a field has multiplicative inverse – i.e., $(\exists y \in G)[x.y = 1]$ – G does not have an inverse operator x^{-1}. We can easily add a unary inverse operator $^{-1}$, or binary division operators $/$ or $\div$, in which case the algebras $G(^{-1})$, $G(/)$ or $G(\div)$ may become partial. Rings and fields so extended are generally referred to as meadows [21,14].

In the case of the rational numbers as a data type for computation, inverse or division are essential. The rings and fields of rationals have only the constants and operations $0, 1, x + y, -x, x.y$ and are not minimal algebras; they become minimal when a multiplicative inverse x^{-1} or a binary division x/y or $x \div y$ is added.

Let Σ_m be the signature of a field $G(\div)$ extended with division as a partial operation; $G(\div)$ is a partial meadow.

In what follows we will endeavour to make division total. If we do this by using an element of G such as 0 then we get an involutive meadow $\mathsf{Tot}_0(G(\div)) = G_0(\div)$ of signature Σ_m.

In particular, if we add an absorbtive element $\perp$ to $G(\div)$ then we get a common meadow $\mathsf{Enl}_\perp(G(\div)) = G_\perp(\div)$ with the signature $\Sigma_{cm} = \Sigma_m \cup \{\perp\}$. We notice that we are adapting and simplifying the terminology of [17] by defining as a common meadow what has been called a common cancellation meadow in [17]. The previous notion of a common meadow is now recovered as: a model of the axioms of Table 1. This adaptation is made in order to make the notion of a common meadow independent of any particular collection of axioms.

From the point of view of algebraic specifications, function names are important and the difference between say $\div$ and $/$ matters. From the perspective of elementary arithmetic, the difference does not seem to matter at all. Not distinguishing function names and the corresponding clustering of synonyms we consider to be an instance of assimilation as proposed

in [41] which may be followed by a phase of disassimilation if one intends to be more precise about names. However, it is noteworthy that division has acquired at least 5 names:

$$x \div y,\ x/y,\ x : y,\ {}^{x}/_{y},\ \frac{x}{y}$$

These have different uses: the first three are inline notations and are poor at expressing nested divisions; the fourth is used only with x, y positive natural numerals; the last is best suited for nesting divisions and complex expressions in general.

Subtle differences concerning the use of familiar names for functions may be helpful. For instance, here we prefer to distinguish $\div$ and $/$ in order to use $\div$ for a division operation for which the preferred interpretation is a partial function: we have a story line where the primary meaning of $\div$ is a partial function that will be compared with various total versions of division usually denoted $/$. To $/$ and $\div$ we also add the bar notation of fractions $\frac{\square}{\square}$, whereas we will not make use of $x : y$ and ${}^{x}/_{y}$.

3. Equational specifications of partial algebras

3.1. Four methods

We formulate four methods for the specification of a partial data type A using a related total data type B. The general algebraic scope of the methods is open, although all apply smoothly to the case of the rationals with division. For each of our methods, we will sketch the technique in general and then apply them to the rationals with division. The total algebra B customised for partial algebra A can be made using either a data type enlargement by new data (external totalisation) or a data type extension by existing data (internal totalisation).

3.1.1. External: adding a unique absorbtive element $\bot$

In this method, we enlarge the data type A by adding a unique new element $\bot \notin A$ that is absorbtive, i.e., if $\bot$ is an argument to an operation of A then its value is $\bot$. The partial operations are extended to return $\bot$ when undefined. This makes the total data type B. In fact, $B = \mathsf{Enl}_{\bot}(A)$.

We give an algebraic specification (Σ', E) of B, if possible (we address the scope of this step in the next section cf. Proposition 4.1).

Then we make the restriction that removes $\bot$ thus obtaining $A = \mathsf{Pdt}_{\bot}(B)$. In fact, the process is invariant (Proposition 2.2): $A = \mathsf{Pdt}_{\bot}(\mathsf{Enl}_{\bot}(A))$. Now we apply this to the rationals.

The rationals with division. Here $B = \mathsf{Enl}_{\bot}(Q(\div))$ is a common meadow of rationals. For common meadows, see section 5.2, and for a new specification of $\mathsf{Enl}_{\bot}(Q(\div))$ see Proposition 5.2 below.

This option is common in pocket calculators to make operators total, where error has the algebraic properties of $\bot$.

3.1.2. External: adding several special elements

The previous method of adding $\bot$ can be (and usually is) more nuanced and so complicated. For example, to make division total in some numerical data types, flags such as ∞ (unsigned) or $+\infty, -\infty$, together with $\bot$ to control the new elements, are added.

To sketch a general method for several special elements, suppose a set $\{c_1, \ldots, c_k\}$ of new elements have been added as constants to A to define a total data type B.

We give an algebraic specification (Σ', E) of B, if possible.

Then we remove the constants to recover A by setting $A = \mathsf{Pdt}_{\{c_1,\ldots,c_k\}}(B)$ for the several constants $\{c_1, \ldots, c_k\}$ at the same time.

This option is common in computer arithmetics where various flags are added to make operators total. Now we apply this to the rationals in two cases.

The rationals with division. Here are two important options inspired by the need for totality in computer arithmetics.

Wheels are data types that add both ∞ (unsigned) and $\bot$ to arithmetic structures [46,29].

We enrich $Q(\div)$ to a wheel $B = \mathsf{Wheel}_{\infty,\bot}(Q(\div))$ of rational numbers. A wheel of rational numbers has an equational specification [23]. Thus, with B a wheel of rationals, we can take: $A = \mathsf{Pdt}_{\infty,\bot}(B) = \mathsf{Pdt}_{\infty}(\mathsf{Pdt}_{\bot}(B)$.

The transreals are data types that add $+\infty, -\infty$ together with $\bot$ to arithmetic structures [2,30]. We enrich $Q(\div)$ to a $B = \mathsf{Trans}_{+\infty,-\infty,\bot}(Q(\div))$. The transrationals have an equational specification [22]. Thus, with B a data type of transrationals: $A = \mathsf{Pdt}_{+\infty,-\infty,\bot}(B)$.

3.1.3. Internal: totalise using existing elements

We need not add new elements to totalise an operation as we can simply use elements from the domain. For a partial algebra A we begin by building a suitable data type B_0 where the partial operations of A have been (i) renamed and then (ii) made total using elements from A. The renaming distinguishes the old, partial operations from the new, total operations.

Then we give an algebraic specification (Σ', E) of B_0, if possible.

Next, we extend B_0 with a 4-place conditional operator $x \lhd u = v \rhd y$. This makes an algebra B_1.

We make a further expansion to B_2 by adding functions with the old names to B_1. These original operations are recovered using explicit definitions of the form: $F(x_1, \ldots, x_n) = t$ where t is a term involving the conditional operator to reintroduce partiality. Finally, we take a reduct of the original signature to remove the additions. Now we apply this to the rationals.

The rationals with division. To make B_0, we rename $\div$ to $/$ and define $x/0 = 0$ (or we may choose any $c \in A$).

This B_0 is an involutive meadow $Q_0(/)$. Now we give an equational specification (Σ', E) of $Q_0(/)$ – this can be done, see subsection 5.1.

On extending this algebra with the conditional operator above we get B_1. In B_1 we can use the conditional to reintroduce partial $\div$ by the explicit definition:

$$x \div y =_{df} \uparrow \lhd y = 0 \rhd x/y$$

where $\uparrow$ is a constant symbol that has no value, i.e., is undefined. (Strictly speaking, this formula should use Kleene equality rather than $=$ to define partial $\div$; we will discuss this later.)

This makes a new algebra B_2.

We may now take a reduct of the specified algebra to forget $/$ and the conditional operator.

3.1.4. Internal: graph surgery on total operations

To define a partial function f by graph surgery we take a total function g, change its graph by taking out some output values, and then re-name the changed function g' as f.

The rationals with division. To make $\div$ we do surgery on a total division operator $/$. For this we use $Q_0(/)$ as we know $Q_0(/)$ has an equational specification (see above).

Now, we cannot simply remove 0 from $Q_0(/)$ as this is needed for computation using the other operations. However, we can remove certain values of x/y.

Let the algebra $Q/0 \uparrow Q_0$ be derived from Q_0 by taking $x/0$ to be undefined for each $x \in Q$. We find

$$Q(\div) = \mathsf{Pdt}_0(\rho_{(/,\div)}(Q/0 \uparrow Q_0)).$$

3.2. Commentary

Let us consider the methods in the case of the partial data type of rationals $A = Q(\div)$.

The first two external methods used established semantical ideas about numerical calculation, namely adding flags.

In 3.1.1 we had to remove only $\perp$ from a common meadow – a data type that models the semantics of calculators.

In 3.1.2 we had to remove ∞ and $\perp$ from a wheel of rationals – a data type that models a semantics of exact numerical computation inspired by the projective line and Riemann sphere.

In 3.1.2, we also had to remove $+\infty, -\infty$ and $\perp$ from a data type of transrationals – a data type that models totality in floating point computation.

The internal methods, in 3.1.3 and 3.1.4, relied on what are called involutive meadows $Q_0(/)$ in which division is written $/$ and $1/0 = 0$. Involutive meadows of rationals have equational specifications (for instance, the specification given in [21] or the simplification of that specification as given in [15]). In [15], too, several notations are proposed for turning an involutive cancellation meadow $Q_0(/)$ into a structure where division or inverse is partial.

Although surgery on the graph of a function $/$ on $Q_0(/)$ is clear from a semantical point of view, we consider the technique to be unattractive from the perspective of algebraic specifications.

All of these data types have equational specifications, though they are not related by homomorphisms. However, from the point of view of the simplicity of the semantical constructions and equational specifications, the technique based on the common meadow will be shown to be the best way of using total data types to specify $Q(\div)$. The order of steps is simple to state and remember: starting with partial A, (i) $\perp$-enlargement to B, (ii) algebraic specification of B, (iii) $\perp$-restriction of B.

4. Computability perspectives

The theory of equational and conditional equational specifications of data types is intimately connected with the theory of computable data types. Indeed, it is not possible to understand the scope and limits of specification methods without turning to computability theory. Fortunately, there are mature theories of computable data types and especially computable rings and fields [48,49].

In the total case, there are many theorems that comprehensively classify the algebraic specification methods. In particular, computable data types have small finite equational specifications under initial algebra semantics if auxiliary operations are allowed (in the single sorted case, 4 equations using 6 auxiliary functions [19]); furthermore, computable data types can be

equationally specified with good term rewriting properties [20]. In the case of partial algebras, the situation is technically more complicated, but computability is no less relevant to our understanding.

Making partial functions total is one thing in algebra but quite another in computability theory, where partiality is a focus of study and affects everything: making a partial computable function total can make it uncomputable. However, it is easy to prove:

Proposition 4.1. *Let A be a partial computable algebra. Then $B = \mathsf{Enl}_{\perp}(A)$ is a computable total algebra if, and only if, each partial operation of A has a computable domain of definition.*

Fortunately, the rational numbers with division are a partial algebra whose only partial operation has a computable domain of definition, namely $Q - \{0\}$.

4.1. Some basic observations

Consider the two external methods where the specification of a data type involved enlargement by new elements, specification and restriction. Suppose the total data type has a finite (conditional) equational initial algebra specification (Σ', E), which may use auxiliary constants and functions. From the point of view of computability, this method is not excessively powerful as the following proposition limits the computational complexity of the resulting total structure.

Definition 4.1. Given A with signature Σ, let $T_\Sigma(A)$ be the set of terms which are defined in A. A subset H of $T_\Sigma(A) \times T_\Sigma(A)$ is *approximately computably enumerable* if $H = (T_\Sigma(A) \times T_\Sigma(A)) \cap J$ with J a computably enumerable subset of $T_\Sigma \times T_\Sigma$.

Recall that a set is *co-computably enumerable* if its complement is computably enumerable.

Proposition 4.2. *For closed terms $t_1, \ldots, t_n$, if $A = \Sigma\Box(\mathsf{Pdt}_{t_1,\ldots,t_n}(I(\Sigma', E)))$ is a reduct then $T_\Sigma(A)$ is co-computably enumerable and $=_A$ is an approximately computably enumerable relation on $T_\Sigma(A)$.*

Proof. A term $t \in T(\Sigma)$ is in $T_\Sigma(A)$ if it is not the case that either t equals one of the terms $t_1, \ldots, t_n$ or t has a subterm which equals one of $t_1, \ldots, t_n$. Now equality in A is computably enumerable, from which it follows with the description just given that $T_\Sigma(A)$ is co-computably enumerable.

Further, it is immediate that equality on $T_\Sigma(A)$ is the intersection of $T_\Sigma(A) \times T_\Sigma(A)$ with a computably enumerable set of pairs (i.e., the pairs t, r for which $t = r$ can be shown from E). □

Now consider the simpler case where the one absorbtive element $\perp$ is added. With the case of the rationals and arithmetic data types in mind, the following observation further limits the complexity of the resulting data type. Now it is assumed that a constant $\perp$ is in Σ'.

Proposition 4.3. *Suppose the reduct $A = \Sigma\Box(\mathsf{Pdt}_{\perp}(T(\Sigma', E)))$ is such that*
(i) $\perp$ denotes an absorptive element in $T(\Sigma', E)$ and
(ii) $A \models 0 \cdot x = 0$ for some constant 0 and function $\cdot$ of Σ.
Then $T_\Sigma(A)$ is computable and $=_A$ is a computably enumerable relation on $T_\Sigma(A)$.

Proof. Let $B \cong T(\Sigma', E)$ so that A is a $\perp$-restriction of B. We have that $t \in T_\Sigma(A)$ if, and only if, $E \vdash 0 \cdot t = 0$. To see this note that if $t \in T_\Sigma(A)$, according to Proposition 4.2, t is not equal to $\perp$ in B and it has also no subterm which is equal to $\perp$ in B. This is because, by assumption (i), $\perp$ is absorptive in B so if a subterm of t equals $\perp$ in B then so does t itself. Thus $t \in T_\Sigma(A) \iff B \not\models t = \perp \iff E \vdash 0 \cdot t = 0$. It follows that $T_\Sigma(A)$ is computably enumerable and, in combination with the observation of Proposition 4.2 that $T_\Sigma(A)$ is co-computably enumerable, it follows that it is computable.

Again using Proposition 4.2, it follows that $=_A$ is an approximately computably enumerable relation on $T_\Sigma(A)$, and $T_\Sigma(A)$ being computable, $=_A$ is a computably enumerable relation. □

4.2. The complexity of data types obtained via totalisation, specification and expansion

Now consider the case that $B = I(\Sigma', E)$ and an expansion B' where B is given by means of a finite number of conditional definitions.

Proposition 4.4. *Let $A = \Sigma \Box B'$. Assume moreover that $A \models 0 \cdot x = 0$ and that there is another closed term (say 1), defined in A such that $A \models 1 \neq 0$. Now:*
(i) it is possible that $T_\Sigma(A)$ is not computably enumerable, and not co-computably enumerable.
(ii) it is possible that $=_A$ is not computably enumerable and that it is not co-computably enumerable.

Proof. To see this, let C be a minimal, total semi-computable and non-computable data type. Then C is expanded to A by $f_1(x,y) = \uparrow \lhd (x = y) \rhd 0$, $f_2(x,y) = 0 \lhd (x = y) \rhd \uparrow$ and $g(x,y) = 1 \lhd (x = y) \rhd 0$. We find that for closed t and r in T_Σ, $f_1(t,r) \in T_\Sigma(A)$ if, and only if, $A \models t \neq r$. It follows immediately that $T_\Sigma(A)$ is *not* computable, and also *not* semi-computable, because its complement is semi-computable.

Moreover, $f_2(t,r) \in T_\Sigma(A)$ if, and only if, $A \models t = r$ from which it follows that $T_\Sigma(A)$ is not co-semicomputable, as otherwise A must be computable which is not the case by assumption.

The same holds for $=_A$, which cannot be co-semicomputable because then it would be computable. Given that $0 =_A g(t,r)$ if, and only, if $A \models t \neq r$, $=_A$ is not semi-computable. □

The $\perp$-enlargement specification method, if it works, yields an outcome very close to what an initial algebra specification will provide.

Proposition 4.5. *Under the conditions of Proposition 4.3: for closed terms t and r:*

$$t \in T_\Sigma(A) \text{ if, and only if, } E \vdash 0 \cdot t = 0$$

and

$$A \models t = r \text{ if, and only if, } E \vdash t = r \text{ and } E \vdash 0 \cdot t = 0 = 0 \cdot r.$$

In the case of the partial data type $Q(\div)$ and any initial algebra specification (Σ, E) of $\mathsf{Enl}_\perp(Q(\div))$ we have that $Q(\div) = \Sigma \,\square\, \mathsf{Enl}_\perp(Q(\div))$, and that Proposition 4.6 applies in this case:

Proposition 4.6. *Under the above conditions, for closed terms t and r in $T_\Sigma(Q(\div))$:*

$$Q(\div) \models t \neq r \text{ if, and only if, } E \vdash (t - r) \div (t - r) = 1.$$

These computability observations further suggest that the $\perp$-enlargement, specification and restriction method is the most straightforward specification method, at least in the case where the target structure is a field of rational numbers equipped with a partial division operator.

If more than one, but finitely many, peripheral elements are used to obtain an enlargement of an algebra of rationals – in order to prepare for an algebraic specification – then similar results concerning computability can be formulated, though no profitable simplification of the specification methods is to be expected. Reasons for enlargement with more than one peripheral element can vary: not only is division made total, but other features of current or conceivable practice can be captured (e.g., as in the case of the arithmetical datatype of transrational numbers).

5. Equational specifications of the total algebras assigned to $Q(\div)$

For the rationals with division, we consider some natural candidates for the total algebra B associated with partial algebra A by the methods of Section 3.1. In each case we are interested in their equational specification.

5.1. Common meadows, wheels, transrationals, and involutive meadows

As we have mentioned in Section 4, *all* total computable data types have equational specifications with decent algebraic and term rewriting properties.[1] The domain, equality and operations of the partial algebra $Q(\div)$ of rational numbers are computable; the division operator $\div$ is computable and undefined only on 0. This means that *if a total algebra constructed by any of the four methods is computable then it has an equational specification*. Whilst this has implications for the scope of the general methods, the rationals are too important to make do with equational specifications generated by general theory: the data types of rational numbers need to have carefully designed equational specifications that are customised, understandable and useful for various purposes.

The first method used $\mathsf{Enl}_\perp(Q(\div))$ in which $1/0 = \perp$. These have been defined and equationally specified in [17,18]. Since this is our best method we study these data types in the next subsection.

The second method used two important data types.

The wheels used $1/0 = \infty$, where ∞ is unsigned and $\perp$ controls the effects of infinity, see [46,29]. We gave an initial algebra specification (Σ_w, E) of an abstract data type of wheels of rationals in [23].

The transrationals used $1/0 = +\infty$, where $+\infty, -\infty$ are added and $\perp$ controls the effects of infinities, see [2,30]. We gave an initial algebra specification (Σ_{tr}, E) of an abstract data type of transrationals in [22].

[1] This is also true of partial data types using certain semantic interpretations [12].

Table 1
E_{cm}: Equations for common meadows in division notation.

$(x+y)+z=x+(y+z)$	(1)
$x+y=y+x$	(2)
$x+0=x$	(3)
$x+(-x)=0\cdot x$	(4)
$x\cdot(y\cdot z)=(x\cdot y)\cdot z$	(5)
$x\cdot y=y\cdot x$	(6)
$1\cdot x=x$	(7)
$x\cdot(y+z)=x\cdot y+x\cdot z$	(8)
$-(-x)=x$	(9)
$x\div y=x\cdot(1\div y)$	(10)
$x\div x=1+0\div x$	(11)
$1\div(x\cdot y)=(1\div x)\cdot(1\div y)$	(12)
$1\div(1+0\cdot x)=1+0\cdot x$	(13)
$1\div 0=\bot$	(14)
$x+\bot=\bot$	(15)

The last two methods used involutive meadows $Q_0(\div)$ in which $1/0=0$, an option studied in [47,1,42]. We have given an initial algebra specification (Σ_m, E) of the abstract data type of involutive meadows of rationals in [21], and a simplification of that specification in [15].

Now we turn to the common meadows.

5.2. Equational specifications of common meadows

Our favoured method is summarised as follows. Let A be a partial algebra. Let $\bot\notin A$ and add it to A. Make $\bot$ absorbtive and the new value of all undefined values of the operations. This makes the total algebra $\mathsf{Enl}_\bot(A)$. Create an equational specification (Σ, E) for $\mathsf{Enl}_\bot(A)$ so that

$$T(\Sigma, E)\cong \mathsf{Enl}_\bot(A).$$

Finally, recover A by

$$A\cong \mathsf{Pdt}_\bot(T(\Sigma, E)).$$

For this method to establish itself in the case of arithmetic data types we need to understand the theory of common meadows.

5.2.1. Commentary on equations for common meadows

The equational axiomatisation (Σ_{cm}, E_{cm}) of the class of common meadows is displayed in Table 1 with some alterations from their first appearance in [17] and in [18].

We have made the following adaptations: (i) instead of **a** we write $\bot$, (ii) writing $x\div y$ for division instead of $\frac{x}{y}$, and moreover (iii) instead of inversive notation ($1\div x$ as a unary function) we use the notation ($x\div y$ as a binary function).

The 'upgrade' of the equations from inverse notation to division notation does not deviate from the pattern given in [15].

We can strengthen equation (11) to a new form $x\cdot(y\div x)=y+0\div x$; and strengthen the equation (12) to a new form $x\div(y\cdot z)=(x\div y)\cdot(1\div z)$ from which we may infer equation (10), i.e., the conventional connection between division and inversive notation: $x\div y=x\cdot(1\div y)$.

5.2.2. Specification of the rationals

The characteristic χ of a ring is the least n such that the n-fold sum $1+1+\ldots+1=0$; if no such n exists then the characteristic $\chi=0$.

Definition 5.1. Denoting the numerals $\underline{n}=1+1+\ldots+1$ (n-times), we define equations for characteristic $\chi=0$ by $E_{\chi=0}=\{\underline{n}\div\underline{n}=1\mid n\in\mathbb{N}\}$.

The following completeness result is shown in [17].

Proposition 5.1. *An equation $t = r$ over the signature of (partial) meadows is valid in all structures of the form $\mathsf{Enl}_\perp(G(\div))$ with G a field of characteristic 0 if, and only if, $E_{cm} + E_{\chi=0} \vdash t = r$.*

Now we have a new equational specification of the common meadow of rationals:

Proposition 5.2. *$(\Sigma_{cm}, E_{cm} + E_{\chi=0})$ constitutes an initial algebra specification of $\mathsf{Enl}_\perp(Q(\div))$.*

Proof. The soundness of the equations follows by inspection; completeness works as follows. Suppose $t = r$ is a closed identity true in $Q(\div)$. By using fracterm flattening (see [17] and Section 7.1) there are flat (i.e., $\div$ free) closed expressions t_1, t_2, r_1, r_2 such that

$$E_{cm} \vdash t = t_1 \div t_2 \text{ and } E_{cm} \vdash r = r_1 \div r_2.$$

The terms t_1, t_2, r_1, r_2 can be shown equal to numerals, where either
(i) both t_2 and r_2 are 0, in which case $t = \perp = r$, or
(ii) both are not equal to 0.
In this latter case, multiplication with $(-1) \div (-1)$ suffices to make both denominators positive. Now we find

$$Q(\div) \models (t_1 \cdot r_2) \div (t_2 \cdot r_2) = t = r = (r_1 \cdot t_2) \div (t_2 \cdot r_2)$$

from which it follows that $Q(\div) \models t_1 \cdot r_2 = r_1 \cdot t_2$. As a valid identity between integers this is provable from E_{cm} so that also $t = r$ is provable from E_{cm}. □

$E_{\chi=0}$ is the logically weakest set of equations which one may add to E_{cm} in order to obtain an initial algebra specification of $\mathsf{Enl}_\perp(Q(\div))$. The specification of Proposition 5.2 is most general in the sense of [7]. In [7] a specification of an abstract data type of common meadows of rationals is called *most general* precisely if all common meadows of characteristic 0 are models of the specification, where a model of a specification making use of auxiliary functions is supposed to be the reduct of a model of the underlying specification involving those auxiliary functions.

A similar observation concerning specifications of involutive meadows of rationals is made in [9]. In [7] it is noticed that a most general initial algebra specification of the involutive meadows of rational numbers must be infinite, while if auxiliary functions are admitted then a finite most general specification can be found. For common meadows of rationals we have:

Proposition 5.3. *If (Σ_{cm}, E) is a most general specification of $\mathsf{Enl}_\perp(Q(\div))$ then it is infinite.*

Proof. Each equation true of all common meadows is derivable from $E_{cm} + E_{\chi=0}$ (by Proposition 5.1). Suppose for a contradiction that E is finite. Now each $e \in E$ is a consequence of a finite subset of E_e of $E_{cm} + E_{\chi=0}$. Let E_f be the finite union of these E_e for all $e \in E$. The finite specification E would be a consequence of the finite set E_f of $E_{cm} + E_{\chi=0}$. Choose $k \in \mathbb{N}$ such that

$$E_f \subseteq E_{cm} + \{\underline{n} \div \underline{n} = 1 \mid n \in \mathbb{N}, 0 < n < k\}.$$

Now let $p > k$ be a prime and consider the prime field of characteristic p equipped with division: $F_p(\div)$ satisfies E_f. Since $\mathsf{Enl}_\perp(Q(\div))$ is the initial algebra of (Σ_{cm}, E), $F_p(\div)$ must be a homomorphic image of $\mathsf{Enl}_\perp(Q(\div))$. This latter state of affairs is impossible as

$$\mathsf{Enl}_\perp(Q(\div)) \models \underline{p} \div \underline{p} = 1 \text{ whereas } F_p(\div) \models \underline{p} \div \underline{p} = \perp.$$

Here is a contradiction so that the assumption regarding the existence of a finite specification can be rejected. □

We do not know if, with the help of auxiliary functions, a finite most general specification of $\mathsf{Enl}_\perp(Q(\div))$ can be found.

6. Equalities

The study of partial equality is technically very involved [3,45]. Since algebraic specifications are built from equations $t = r$, equality $=$ is dominant but it need not mean something simple. Here we look at 7 equalities that are relevant for working with partial arithmetical data types.

6.1. Equalities derived from total algebras

In the absence of a standard or even a 'most plausible' notion of equality for partial algebras, it is practical to take $G(\div)$, with G a field, as a platform for surveying various notions of partial equality for fields equipped with a partial division operation; $G(\div)$ is a partial meadow.

On total algebras, standard equality presents no problems for identifying elements or syntactic expressions. The constructions of Section 3.1 offer various total semantic models B of partiality in A from which we can simply define a 'partial' equality on A using the standard equality on B. This gives us 4 equalities. In what follows we restrict to arithmetical algebras and let σ be a valuation of variables.

First, we use standard equality $=$ in the common meadow, wheel and transfield:

Definition 6.1. For a common meadow,

$$G(\div), \sigma \models t =_{cm} r \text{ if, and only if, } \mathsf{Enl}_{\perp}(G(\div)), \sigma \models t = r.$$

Definition 6.2. For a wheel,

$$G(\div), \sigma \models t =_{w} r \text{ if, and only if, } \mathsf{Wheel}_{\infty,\perp}(G(\div)), \sigma \models t = r.$$

Definition 6.3. For a transfield,

$$G(\div), \sigma \models t =_{tr} r \text{ if, and only if, } \mathsf{Trans}_{+\infty,-\infty,\perp}(G(\div)), \sigma \models t = r.$$

Further, we can use equality in an involutive meadow and other internal options.

Definition 6.4. For an involutive meadow,

$$G(\div), \sigma \models t =_{im} r \text{ if, and only if, } \mathsf{Tot}_{0}(G(\div)), \sigma \models t = r.$$

Moreover, more generally, for an arbitrary $g \in G$: the equality relation $=_g$ is defined by:

$$G(\div), \sigma \models t =_{g} r \text{ if, and only if, } \mathsf{Tot}_{g}(G(\div)), \sigma \models t = r.$$

We notice that $=_{im}$ is a synonym for $=_0$. When $=_{im}$ is adopted – such as in a division by zero calculus (e.g., [38,43]) and in the theory of involutive meadows (e.g., [42,16]) – for simplicity $=$ is written instead of $=_{im}$ or $=_0$ (or any other ad hoc notation for an equality relation).

6.2. Kleene equality for $G(\div)$

Kleene equality logic works over an arbitrary partial algebra [36]. The idea of Kleene equality $\simeq$ is that it works on closed Σ expressions as in $\mathsf{Enl}_{\perp}(A)$ while for open equations universal quantification ranges over non-$\perp$ elements of A only.

Proposition 6.1. *Let G be an arbitrary field with signature Σ and extend to the meadow signature $\Sigma_m = \Sigma \cup \{\div\}$. Let $t, r \in T(\Sigma_m)$ be such that the free variables of t, r are contained in $x_1, \ldots, x_k$, then:*

$$G(\div) \models t \simeq r \iff \mathsf{Enl}_{\perp}(G(\div)) \models (\bigwedge_{i=1,\ldots,k} 0 \cdot x_i = 0) \to t = r.$$

Proof. Immediate from the definition of $\simeq$. □

Proposition 6.2. *For terms t, r over the signature of meadows Σ_m with variables among $x_1, \ldots, x_k$ the following are equivalent:*

1. *$t \simeq r$ is valid in all partial meadows $G(\div)$ of characteristic 0 (i.e., all $G(\div)$ with G a field of characteristic 0);*
2. *$E_{cm} + E_{\chi=0} \vdash (\bigwedge_{i=1,\ldots,k} 0 \cdot x_i = 0) \to t = r$.*

Proof. The 'if' part is immediate from the soundness of the axioms and the proof system $\vdash$. The 'only' if part works as follows: If $t = r$ is true in all partial meadows with characteristic 0 then with Proposition 6.1 $(\bigwedge_{i=1,\ldots,k} 0 \cdot x_i = 0) \to t = r$ holds in all common meadows with characteristic 0. From this it follows that

$$(\sum_{i=1}^{n} 0 \cdot x_i) + t = (\sum_{i=1}^{n} 0 \cdot x_i) + r$$

holds in all these structures. Then Proposition 5.1 yields

$$E_{cm} + E_{\chi=0} \vdash (\sum_{i=1}^{n} 0 \cdot x_i) + t = (\sum_{i=1}^{n} 0 \cdot x_i) + r$$

from which one obtains

$$E_{cm} + E_{\chi=0} \vdash (\bigwedge_{i=1,\ldots,k} 0 \cdot x_i = 0) \to t = r$$

by means of equational logic. □

6.3. NaN-equality for $G(\div)$

In computer arithmetic it is not unusual (but also not universally accepted) to refer to an entity outside the conventional number system as a NaN, denoting 'not a number'. In addition, it is often assumed that an NaN cannot be equal to any other entity, not even to itself. In the context of choosing an arbitrary field G and adding division, $G(\div)$, this idea leads to a different notion of equality which we refer to as *NaN equality*:

Definition 6.5. $G(\div), \sigma \models t \simeq_{\mathsf{f}} r \iff G(\div), \sigma \models t \simeq r$ and $G(\div), \sigma \models 0 \cdot t \simeq 0$.

6.4. 3-Valued equality

A seventh notion of equality plays an important role in understanding partiality [26]. Here satisfaction relation $G(\div), \sigma \models t \simeq_{\mathsf{d}} r$ yields a result in a three-valued logic with truth values t, f and d.

Definition 6.6. There are three cases:
$(G(\div), \sigma \models t \simeq_{\mathsf{d}} r) = \mathsf{t}$ if, and only if, $G(\div), \sigma \models t \simeq_{\mathsf{f}} r$,
$(G(\div), \sigma \models t \simeq_{\mathsf{d}} r) = \mathsf{d}$ if either $G(\div), \sigma \models t = 1 \div 0$ or $(G(\div), \sigma \models r = 1 \div 0)$ (or both),
$(G(\div), \sigma \models t \simeq_{\mathsf{d}} r) = \mathsf{f}$ if $G(\div), \sigma \models t \neq 1 \div 0$ and $G(\div), \sigma \models r \neq 1 \div 0$ and $G(\div), \sigma \models t \neq r$.

Thus, in other words: $t \simeq_{\mathsf{d}} r$ yields t if both $t \simeq r$ and $t \simeq_{\mathsf{f}} r$ are true, and yields f if either $t \simeq r$ or $t \simeq_{\mathsf{f}} r$ is false, while yielding d otherwise.

6.5. The equality sign in arithmetical practice

The equality sign $=$ is ubiquitous in arithmetical practice. For instance, in teaching $=$ must be understood as a key component of an informal language governed by a collection of informal conventions. It is not plausible to expect that the informal conventions at work in some practical arithmetical context will coincide precisely with the formal prescriptions of an equality relation chosen from

$$=_{cm}, =_{w}, =_{tr}, =_{im}, \simeq, \simeq_{\mathsf{f}}, \simeq_{\mathsf{d}} .$$

Actually, the equality sign $=$ as used in daily practice in arithmetic is some mix of these 7 options (and others), where the particularities of the mix may differ depending on the context at hand. Thus, it is a good example of an *assimilation* in the terminology of [41]. However, we expect that a thorough analysis of $=$ as used in practice will involve specialised syntactic and semantic studies creating the study of legal/illegal arithmetical texts.

To the best of our knowledge there is no generally agreed notion of equality which stands out as the most plausible choice for an interpretation of the equality sign on $G(\div)$. Here we take into account that in practice: (i) the equality sign is used as a relation between expressions rather than between mere values, and (ii) the idea that the semantics of the equality sign can be determined independently of the presence of $\perp$ and without taking expressions into account is unwarranted.

For $\mathsf{Enl}_{\perp}(G(\div))$ the situation is different, and in fact straightforward, and $=$ as used in its first order logic stands out as the most plausible interpretation of the equality sign in that setting.

We notice that the conventional practice of arithmetic considers the occurrence of certain expressions, notably $\frac{1}{0}$, as undesirable. We propose to introduce a notion of legality to label assertions or texts concerning elementary arithmetic which are not rejected in conventional practice, while qualifying an assertion non-legal if it is plausibly rejected. Providing an unambiguous definition of legality may not be possible, as opinions regarding acceptance and rejection of texts may diverge, in which case a spectrum of notions of legality may come about. We will, however, indicate examples of non-legal assertions which we consider to be undisputed. For a term t we define t to be *not legal* if for each valuation σ, $\mathsf{Enl}_{\perp}(Q(\div)), \sigma \models t = \perp$. Thus a legal expression must allow taking a non-$\perp$ value. We then propose that an assertion involving a non-legal expression is itself non-legal.

Table 2
E_{cfc}: Equations of common fracterm calculus in conventional notation.

$$(x+y)+z = x+(y+z)$$
$$x+y = y+x$$
$$x+0 = x$$
$$x+(-x) = 0\cdot x$$
$$x\cdot(y\cdot z) = (x\cdot y)\cdot z$$
$$x\cdot y = y\cdot x$$
$$1\cdot x = x$$
$$x\cdot(y+z) = x\cdot y + x\cdot z$$
$$-(-x) = x$$
$$\frac{x}{y} = x\cdot\frac{1}{y}$$
$$\frac{x}{x} = 1+\frac{0}{x}$$
$$\frac{1}{x\cdot y} = \frac{1}{x}\cdot\frac{1}{y}$$
$$\frac{1}{1+0\cdot x} = 1+0\cdot x$$
$$\frac{1}{0} = \bot$$
$$x+\bot = \bot$$

For instance $\phi \equiv \mathsf{false} \to \frac{1}{0} = \frac{1}{0}$ is true in $\mathsf{Enl}_{\bot}(Q(\div))$ and which may be considered true in $Q(\div)$ for that reason. Nevertheless ϕ may be considered non-legal because it contains a non-legal subterm. Similarly $\phi \equiv \mathsf{false} \to \frac{1}{x-x} = 1$ is true in $\mathsf{Enl}_{\bot}(Q(\div))$, as well as in $Q(\div)$, while not legal.

Then consider $(x \neq 0 \wedge x \neq \bot) \to \frac{x}{x} = 1$ which is satisfied and unproblematic in $\mathsf{Enl}_{\bot}(Q(\div))$ so that $x \neq 0 \to \frac{x}{x} = 1$ can be considered valid in $Q(\div)$. Upon substituting $x = 0$, however, the situation changes and $0 \neq 0 \to \frac{0}{0} = 1$ may be considered non-legal because it contains a non-legal subterm.

7. Fracterm calculi

The use of a common meadow $\mathsf{Enl}_{\bot}(G(\div))$ to provide a stable workable notion of $=$ for the special case of elementary arithmetic with division may have wider scope and significance than being a mere example of the application of the theory of abstract data types. Instead of thinking about equational specifications of abstract data types, consider thinking about a useable set of laws for manipulating the standard arithmetical forms of fractions. The arithmetical fractions in elementary teaching can be faithfully modelled by syntax, and their algebra by structures corresponding with $G(\div)$. To reproduce fractions, the notation $x \div y$ is replaced by $\frac{x}{y}$ and these syntactic expressions, as well as substitution instances thereof, we call fracterms.

7.1. A fracterm calculus for common meadows

The axioms for common meadows in Table 2 lay down equational laws for division using the horizontal bar notation for fracterms. The change of notation makes explicit a change of representation through syntax: the ill-defined notion of fraction becomes clear in the definition of fracterm.

With the change of notation, and the introduction of formal syntax of fracterms, these axioms may alternatively be restyled as the *fracterm calculus of common meadows*.

What we will refer to as common fracterm calculus is a calculus specific for common meadows. Common fracterm calculus is merely another name for the specification of common meadows in division notation. We propose that the default interpretation of 'fracterm calculus' is common fracterm calculus, thereby expressing our belief that the common fracterm calculus is the most convincing instance of a fracterm calculus.

A central first property of any calculus formalising fractions is the possibility of flattening fracterms:

Definition 7.1. A fracterm t is *flat* if it is of the form $t = \frac{p}{q}$ where the terms p and q do not contain division.

Theorem 7.1. *Any fracterm t can be reduced by the laws of Table 2 to a flat fracterm $\frac{p}{q}$.*

Remarkably the fracterm flattening property cannot be obtained without extending the domain of rational numbers.

Theorem 7.2. *If a total enlargement A of $Q(\div)$ has no peripheral numbers, i.e., no elements outside the rationals occur in the domain, so that A is of the form $\mathsf{Tot}_t(Q(\div))$ with t a closed term of the form $\frac{n}{m}$ with $m > 0$, then the corresponding fracterm calculus does not enjoy the flattening property.*

Proof. Suppose that the total enlargement A of $Q(-)$ without peripheral numbers, allows fracterm flattening. We consider the following expression:

$$h(x, y, u, v) \equiv \frac{x}{y} + \frac{u}{v}.$$

Let p and q be flat (i.e., division free) expressions such that $A \models h(x, y, u, v) = \frac{p}{q}$. We notice that p and q are polynomials with integer coefficients. We first show that y is a factor of q, and with a similar proof that v is a factor of q so that $q = y \cdot v \cdot q'$, for some polynomial q' with integer coefficients. To see that y is a factor of q we write $q = q_1 \cdot y + q_2$ where y does not occur in q_2. Suppose that $q_2(x, u, v)$ is nonzero then for some $a, b, c \in Q$, $q_2(a, b, c) \neq 0$. Thus $\frac{p}{q}$ is a continuous function of x sufficiently close to $(x = a, y = 0, u = b, v = c)$ which cannot be the case given the definition of h.

Next we notice that $p(x, y, z, u) = (x \cdot v + y \cdot u) \cdot q'(x, y, z, u)$ for all x, y, z, u where h is continuous. It follows that $p(x, y, z, u) = (x \cdot v + y \cdot u) \cdot q'(x, y, z, u)$ everywhere. Now choosing $y = u = v = 0$ one finds

$$A \models \frac{x}{0} + \frac{0}{0} = \frac{(x \cdot 0 + 0 \cdot 0) \cdot q'(x, 0, 0, 0)}{0 \cdot 0 \cdot q'(x, 0, 0, 0)} = \frac{0}{0}.$$

Because there are no peripherals $\frac{x}{0} \in Q$ for all $x \in Q$ (including 0). It follows that for all x, $\frac{x}{0} = 0$, i.e., A is an involutive common meadow for which it is known from [16] that it does not allow flattening. □

Theorem 7.2 complements the classification results on flattening with peripheral numbers in [25]. Some technical highlights of fracterm calculus are:

- terminology for and classification of fracterms [6];
- transformation of arbitrary fracterms into flat fracterms [17];
- logical complexity for fracterm calculi without flattening [5]; and
- analysis of the informal conventions of practical arithmetic about the legality of texts with division by zero.

7.2. Suppes-Ono fracterm calculus

Just as we can propose a fracterm calculus based on common meadows, we can formulate one for each other 'competing' equational specification. Thus, besides common fracterm calculus we propose *Suppes-Ono fracterm calculus* as a name for the specification in Table 3. E_{sofc} of Table 3 is just the specification of involutive meadows in division notation as given in [15]. The name Suppes-Ono fracterm calculus for the calculus of Table 3 (which occurs in [15] already) is chosen for the following reasons:

(i) Suppes in [47] clearly expressed for the first time (to the best of our knowledge) that $\frac{1}{0} = 0$ constitutes a possible and satisfactory solution of a meaningful problem (what is the result of division by zero) which admits a plurality of solutions each of which come with advantages and with disadvantages (see also [1]), and

(ii) Ono in [42] pioneered what happens to the equational logic of fields upon adopting the equation $\frac{x}{0} = 0$.

Regarding flattening for Suppes-Ono fracterm calculus, it is shown in [11] that each fracterm can be written as a *sum of flat fracterms*. We refer to this property of the Suppes-Ono fracterm calculus as the *quasi-flattening* property for fracterms.

Moreover, in [16], it is shown that arbitrarily long sums are needed for that objective, and consequently Suppes-Ono fracterm calculus does not allow fracterm flattening proper. In the case of expressions having a single free variable only it is shown in [10] that all such expressions can be written in mixed fracterm form, being the sum of a polynomial and a flat fracterm.

For $\simeq$, $\simeq_{\mathsf{f}}$ and $\simeq_{\mathsf{d}}$ we are unaware of relevant existing literature, and corresponding equational specifications have yet to be developed. For proof systems for Kleene equality in general we refer to [45,3].

8. Concluding remarks

8.1. Reflection

Of the four methods for specifying partial data type A, the method of enlarging by a single absorbtive element $\perp$ is most appealing. Semantically, the total algebra B is the simplest to explain and remember and has good computability properties that give some indication of the scope of the general method (e.g., Proposition 4.1).

Table 3
E_{sofc}: A fracterm calculus for Suppes-Ono.

$$(x+y)+z=x+(y+z)$$
$$x+y=y+x$$
$$x+0=x$$
$$x+(-x)=0$$
$$x\cdot(y\cdot z)=(x\cdot y)\cdot z$$
$$x\cdot y=y\cdot x$$
$$1\cdot x=x$$
$$x\cdot(y+z)=x\cdot y+x\cdot z$$
$$-(-x)=x$$
$$\frac{x}{y}=x\cdot\frac{1}{y}$$
$$\frac{1}{(\frac{1}{x})}=x$$
$$\frac{x\cdot x}{x}=x$$

In the case of arithmetical structures like the rationals, the method using common meadows that have equational specifications that are recognisable, informative and useful. Thus, from the perspective of abstract data type specifications and reasoning, when contemplating $Q(\div)$ as a partial abstract data type, this paper establishes that the corresponding abstract data type of common meadows of rationals $\mathsf{Enl}_{\perp}(Q(\div))$ will play a central role. This role is enhanced by the extent that we use the partial equality $=_{cm}$ in connection with $Q(\div)$, based on the native equality in $\mathsf{Enl}_{\perp}(Q(\div))$.

8.2. Partial algebraic specifications

For reference let us recall some partial specification methods for partial data types. In programming, partial operations arise easily and the problem of specifying them have stimulated many technical ideas, such as guards against applying operations on data outside their domains – definedness predicates in conditional axioms [27], subsorting [33], guarded algebras [34]. Partiality is an issue in both specification and verification [35]. There are comprehensive surveys [39] and substantial mathematical monographs [28,44] to chart the research programme. A measure of the state of the art is to be found in the design decisions of the specification language CASL [40]. In CASL, basic specifications define a class of many-sorted partial first order structures. One method for totalisation that can be tried is designating the domain of a partial operation as a subsort. Indeed, a specification of the rationals using subsorts is to be found in Maude. Of course, the theory of partial specifications aims at being generally applicable. But such advanced methods have a cost and it is a high one when specifying data types such as the rational numbers with division $Q(\div)$.

Let us consider the early basic method offered by Broy and Wirsing in [27]. To specify a partial data type A of signature Σ, they add a predicate $D(-)$ to denote the definedness of a term t. Then assertions $D(t)$ can be placed in assumptions and conclusions of conditional axioms in which the predicates occur positively. Under these circumstances they show that initial algebras exist and can be used to specify A up to isomorphism. In fact, the notion of equality they use is what we call Kleene equality.

Using the method of definedness predicates, one can provide an initial partial algebra specification of $Q(\div)$. However, the specification is logically complicated as it makes use of predicates, conditional formulae, and Kleene equality – this makes it quite far removed from the elegant and efficient theories of equational specifications such as Table 1 and its sibling Table 2.

Thus, we focus on specification methods which use algebraic specifications for the specification of a total abstract data type as an intermediate stage, followed by a second stage which leads to the required partial abstract data type. The resulting algebras are structures which come without a particular (favoured) equality relation, and in fact several plausible equalities can then be introduced for various purposes.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] J.A. Anderson, J.A. Bergstra, Review of Suppes 1957 proposals for division by zero, in: Transmathematica, 2021, https://doi.org/10.36285/tm.53.

[2] J.A. Anderson, N. Völker, A.A. Adams, Perspecx Machine VIII, axioms of transreal arithmetic, in: J. Latecki, D.M. Mount, A.Y. Wu (Eds.), Vision Geometry XV, in: Proc. SPIE, vol. 6499, 2007, 649902.
[3] H. Andreka, W. Craig, I. Nemeti, A system of logic for partial functions under existence-dependent Kleene equality, J. Symb. Log. 53 (3) (1988) 834–839.
[4] J.A. Bergstra, Division by zero, a survey of options, in: Transmathematica, 2019, https://doi.org/10.36285/tm.v0i0.17.
[5] J.A. Bergstra, Fractions in transrational arithmetic, in: Transmathematica, 2020, https://doi.org/10.36285/tm.19.
[6] J.A. Bergstra, Arithmetical data types, fracterms, and the fraction definition problem, in: Transmathematica, 2020, https://doi.org/10.36285/tm.33.
[7] J.A. Bergstra, Most general algebraic specifications for an abstract data type of rational numbers, Sci. Ann. Comput. Sci. 30 (1) (2020) 1–24, https://doi.org/10.7561/SACS.2020.1.1.
[8] J.A. Bergstra, Division by zero in logic and computing, HAL archives ouvertes, https://hal.archives-ouvertes.fr/hal-03184956, 2021.
[9] J.A. Bergstra, I. Bethke, Subvarieties of the variety of meadows, Sci. Ann. Comput. Sci. 27 (1) (2017) 1–18, https://doi.org/10.7561/SACS.2017.1.1.
[10] J.A. Bergstra, I. Bethke, D. Hendriks, Universality of mixed fractions in divisive meadows, https://arxiv.org/abs/1707.00499, 2017.
[11] J.A. Bergstra, I. Bethke, A. Ponse, Cancellation meadows: a generic basis theorem and some applications, Comput. J. 56 (1) (2013) 3–14, https://doi.org/10.1093/comjnl/bxs028, Also http://arxiv.org/abs/0803.3969.
[12] J.A. Bergstra, M. Broy, J.V. Tucker, M. Wirsing, On the power of algebraic specifications, in: J. Gruska, M. Chytil (Eds.), Mathematical Foundations of Computer Science 1981, MFCS 1981, in: Lecture Notes in Computer Science, vol. 118, Springer, 1981.
[13] J.A. Bergstra, J. Heering, P. Klint, Module algebra, J. ACM 37 (2) (1990) 335–372.
[14] J.A. Bergstra, Y. Hirshfeld, J.V. Tucker, Meadows and the equational specification of division, Theor. Comput. Sci. 410 (12) (2009) 1261–1271.
[15] J.A. Bergstra, C.A. Middelburg, Inversive meadows and divisive meadows, J. Appl. Log. 9 (3) (2011) 203–220, Also https://arxiv.org/abs/0907.0540.
[16] J.A. Bergstra, C.A. Middelburg, Transformation of fractions into simple fractions in divisive meadows, J. Appl. Log. 16 (2015) 92–110, Also https://arxiv.org/abs/1510.06233.
[17] J.A. Bergstra, A. Ponse, Division by zero in common meadows, in: R. de Nicola, R. Hennicker (Eds.), Software, Services, and Systems (Wirsing Festschrift), in: Lecture Notes in Computer Science, vol. 8950, Springer, 2015, pp. 46–61, Available in improved form at https://arxiv.org/abs/1406.6878v4.
[18] J.A. Bergstra, A. Ponse, Fracpairs and fractions over a reduced commutative ring, Indag. Math. 27 (2016) 727–748, Also https://arxiv.org/abs/1411.4410.
[19] J.A. Bergstra, J.V. Tucker, The completeness of the algebraic specification methods for computable data types, Inf. Control 54 (3) (1982) 186–200.
[20] J.A. Bergstra, J.V. Tucker, Equational specifications, complete term rewriting systems, and computable and semicomputable algebras, J. ACM 42 (6) (1995) 1194–1230.
[21] J.A. Bergstra, J.V. Tucker, The rational numbers as an abstract data type, J. ACM 54 (2) (2007) 7, https://doi.org/10.1145/1219092.1219095.
[22] J.A. Bergstra, J.V. Tucker, The transrational numbers as an abstract data type, in: Transmathematica, 2020, https://doi.org/10.36285/tm.47.
[23] J.A. Bergstra, J.V. Tucker, The wheel of rational numbers as an abstract data type, in: M. Roggenbach (Ed.), Recent Trends in Algebraic Development Techniques, WADT 2020, in: Lecture Notes in Computer Science, vol. 12669, Springer, 2021, pp. 13–30.
[24] J.A. Bergstra, J.V. Tucker, Totalising partial algebras: teams and splinters, in: Transmathematica, 2022, https://doi.org/10.36285/tm.57.
[25] J.A. Bergstra, J.V. Tucker, Which arithmetical data types admit fracterm flattening?, Sci. Ann. Comput. Sci. 32 (1) (2022) 87–107, https://doi.org/10.7561/SACS.2022.1.87.
[26] J.A. Bergstra, I. Bethke, P.H. Rodenburg, A propositional logic with 4 values: true, false, divergent and meaningless, J. Appl. Non-Class. Log. 5 (2) (1995) 199–217.
[27] M. Broy, M. Wirsing, Partial abstract types, Acta Inform. 18 (1982) 47–64.
[28] P. Burmeister, A Model Theoretic Oriented Approach to Partial Algebras, Akademie-Verlag, 1986.
[29] J. Carlström, Wheels – On division by zero, Math. Struct. Comput. Sci. 14 (1) (2004) 143–184, https://doi.org/10.1017/S0960129503004110.
[30] T.S. dos Reis, W. Gomide, J.A.D.W. Anderson, Construction of the transreal numbers and algebraic transfields, IAENG Int. J. Appl. Math. 46 (1) (2016) 11–23, http://www.iaeng.org/IJAM/issues_v46/issue_1/IJAM_46_1_03.pdf.
[31] H.D. Ehrich, M. Wolf, J. Loeckx, Specification of Abstract Data Types, Vieweg Teubner, 1997.
[32] H. Ehrig, B. Mahr, Fundamentals of Algebraic Specification 1: Equations and Initial Semantics, EATCS Monographs on Theoretical Computer Science, vol. 6, Springer, 1985.
[33] J. Goguen, R. Diaconescu, An Oxford survey of order sorted algebra, Math. Struct. Comput. Sci. 4 (3) (1994) 363–392.
[34] M. Haveraaen, E.G. Wagner, Guarded algebras: disguising partiality so you won't know whether it's there, in: D. Bert, C. Choppy, P.D. Mosses (Eds.), Recent Trends in Algebraic Development Techniques, in: Lecture Notes in Computer Science, vol. 1827, Springer, 1999, pp. 182–200.
[35] C.B. Jones, C.A. Middelburg, A typed logic of partial functions, reconstructed classically, Acta Inform. 31 (1994) 399–430.
[36] S.C. Kleene, Introduction to Metamathematics, North-Holland, 1952.
[37] K. Meinke, J.V. Tucker, Universal algebra, in: S. Abramsky, D. Gabbay, T. Maibaum (Eds.), Handbook of Logic for Computer Science, Oxford University Press, 1992, pp. 189–411.
[38] H. Michiwaki, S. Saitoh, N. Yamada, Reality of the division by zero $z/0 = 0$, Int. J. Appl. Phys. Math. (2016), https://doi.org/10.17706/ijapm.2016.6.1.1-8.
[39] P.D. Mosses, The use of sorts in algebraic data type specification, in: Michel Bidoit, Christine Choppy (Eds.), Recent Trends in Data Type Specification, in: Lecture Notes in Computer Science, vol. 655, Springer, 1993, pp. 66–91.
[40] P.D. Mosses (Ed.), CASL Reference Manual. The Complete Documentation of the Common Algebraic Specification Language, Lecture Notes in Computer Science, vol. 2960, Springer, 2004.
[41] J-F. Nicaud, D. Bouhineau, J-M. Gelis, Syntax and semantics in algebra, in: Proc. 12th ICMI Study Conference, The University of Melbourne, 2001, HAL archives-ouvertes, https://hal.archives-ouvertes.fr/hal-00962023/document, 2001.
[42] H. Ono, Equational theories and universal theories of fields, J. Math. Soc. Jpn. 35 (2) (1983) 289–306, https://doi.org/10.2969/jmsj/03520289.
[43] H. Okumura, S. Saitoh, Applications of the division by zero calculus to Wasan geometry, Glob. J. Adv. Res. Classical Mod. Geom. 7 (2) (2018) 44–49.
[44] H. Reichel, Initial Computability, Algebraic Specifications and Partial Algebras, Oxford Science Publications, 1987.
[45] A. Robinson, Equational logic of partial functions under Kleene equality: a complete and an incomplete set of rules, J. Symb. Log. 54 (2) (1989) 354–362.
[46] A. Setzer, Wheels, Unpublished draft, http://www.cs.swan.ac.uk/csetzer/articles/wheel.pdf, 1997.
[47] P. Suppes, Introduction to Logic, Van Nostrand Reinhold Company, 1957, http://web.mit.edu/gleitz/www/Introduction%20to%20Logic%20-%20P.%20Suppes%20(1957)%20WW.pdf.
[48] V. Stoltenberg-Hansen, J.V. Tucker, Effective algebras, in: S. Abramsky, D. Gabbay, T. Maibaum (Eds.), Handbook of Logic in Computer Science. Volume IV: Semantic Modelling, Oxford University Press, 1995, pp. 357–526.
[49] V. Stoltenberg-Hansen, J.V. Tucker, Computable rings and fields, in: E. Griffor (Ed.), Handbook of Computability Theory, Elsevier, 1999, pp. 363–447.
[50] W. Wechler, Universal Algebra for Computer Scientists, Springer-Verlag, 1992.