University of Amsterdam

Theory of Computer Science

Four Complete Datatype Defining Rewrite Systems for an Abstract Datatype of Natural Numbers (version 2)

J.A. Bergstra

Report TCS1407v2 June 2014

J.A. Bergstra

section Theory of Computer Science
Faculty of Science
University of Amsterdam

Science Park 904
1098 XH Amsterdam
the Netherlands

tel. +31 20 525.7591
e-mail: J.A.Bergstra@uva.nl

Theory of Computer Science Electronic Report Series

Four Complete Datatype Defining Rewrite Systems for an Abstract Datatype of Natural Numbers
(version 2*)

Jan A. Bergstra

Informatics Institute, University of Amsterdam
email: j.a.bergstra@uva.nl

Abstract

Natural numbers with zero, one, successor, addition and multiplication, constitute a classic example of an abstract datatype amenable for equational initial algebra specification. Datatype defining rewrite systems provide a specification which at the same time is a complete, that is confluent and strongly terminating, rewrite system thereby providing means for automatic implementation. Syntax for unary, binary, decimal, and hexadecimal notation is introduced and corresponding rewrite systems are designed.

Keywords and phrases: Equational specification, initial algebra, term rewriting system, abstract data type, datatype defining rewrite system.

Contents

*Equations 47, 49, 70, and 72 have been added to the respective tables.

1 Introduction

I will present four algebraic specifications of the same abstract datatype of natural numbers. Each of these specifications comprises a complete, that is ground confluent and strongly terminating, rewriting system. A complete rewriting system serving as a specification for an abstract data type will be called a Complete Datatype Defining Rewrite System (CDDRS).

The concept of datatype is understood as being more specific than that of an abstract datatype in the following sense: in a datatype entities are represented uniquely by preferred closed expressions (normal forms) whereas an abstract datatype, being an isomorphism class of algebras, entities are represented by classes of expressions.

These specifications incorporate different views on the same abstract data type. The unary view provides a term rewriting system where terms in unary notation serve as normal forms. The unary view also provides a semantic specification of binary notation, of decimal notation, and of hexadecimal notation. The three logarithmic notations were modified w.r.t. to conventional notations in such a way that syntactic confusion between these notations cannot arise.

In the binary view normal forms are natural number expressions in binary notation. Having binary expressions as normal forms excludes having terms in unary notation, in decimal notation, or in hexadecimal notation as normal forms.

I also specify the natural numbers in decimal view with a CDDRS having normal forms in decimal notation, and finally a CDDRS is give for a hexadecimal view on natural numbers.

1.1 Motivation

The motivation for this work is that it can serve as a basis for further development in a variety of directions. The following objectives can be mentioned:

- Development of arithmetical datatypes for integers, fractions, and rationals. I
- Development of a precise definition of fractions, a seemingly well-know notion, that is mostly dealt with in an unprecise manner. Such a definition must unavoidably be based on an approach to natural numbers and integers.
- Development of the foundations of an initial part of (potentially innovative) teaching methods for elementary mathematics based on term rewriting and datatypes.
- Philosophical and pragmatic analysis of the concept of a natural number.
- Conceptual analysis of the notion of a secret key (understood as a natural number).

1.2 Notational conventions

Digits are $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E$ and F. Digits are enumerated in precisely this order. The ordering is: $0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < A < B < C < D < E < F$ For the digits $0, 1, 2, 3, 4, 5, 6, 7, 8, A, B, C, D, E$ we denote with i' the successor digit of i in the given enumeration. This is specified formally in Table 1.

Further $\bigwedge_{i=n}^{i=m}[t = r]$, with $n, m \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ and $n = m$ or $n < m$. represents the set of rewrite rules $t = r$ with i instantiated from n to and including m.

1.3 A signature for natural numbers

The signature $\Sigma_{\mathbb{N}}$ has the following elements:

1. sort $\mathbb{N}$,
2. constants 0 and 1,
3. two place functions $+, \cdot : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$,
4. (for unary notation:) a one place function $S : \mathbb{N} \to \mathbb{N}$,
5. (for binary notation:) two one place functions $- \ominus 0, - \ominus 1 : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$,
6. (for decimal notation:) ten one-place functions $- \oslash 0, ..., - \oslash 9 : \mathbb{N} \to \mathbb{N}$,
7. (for hexadecimal notation:) sixteen one-place functions $- \otimes 0, ..., - \otimes F : \mathbb{N} \to \mathbb{N}$.

Unary number terms have the form: $0, S(0), S(S(0)), ..$, for such terms brackets are often omitted: $0, S0, SS0, ...$ Binary number terms (intended normal forms) have the form: $0, 1, 1 \ominus 0, 1 \ominus 1, (1 \ominus 0) \ominus 0....$ In binary number terms like $((1 \ominus 0) \ominus 0) \ominus 1$ brackets and append operators are usually omitted, thus obtaining 1001 as a representation of the (decimal) number 9.

In decimal number terms like $((9 \oslash 7) \oslash 5) \oslash 0$ brackets and composition tokens are usually omitted, thus obtaining 9750 with its usual meaning. Similarly in hexadecimal notation

$((3 \otimes B) \otimes F) \otimes 7$ will be abbreviated to $3BF7$. When needed disambiguation between abbreviated versions of binary, decimal, and hexadecimal notation may be realized by means of subscripts. I refer to [11] for an account of natural number notations and their disambiguation.

1.4 About abstract data types and equational specifications

A survey of equational algebraic specifications for abstract data types is provided in [12]. In [7] one finds the general result that computable data types can be specified by means of specifications which are confluent and strongly terminating term rewriting systems. In other words: for every computable abstract datatype there exists a complete datatype defining rewrite system (CDDRS). The result makes use of the possibility to introduce auxiliary functions.

Some general theory on algebraic specifications is found in [9, 6, 10]. An example of equational specification in the setting of many-valued logics is given in [1], and several more recent applications of equational specifications are found in [**?**, 4, 5, 8].

1.5 RNNs and PRNNs

The predicate (property) of an entity of its being a natural number is supposed to be applicable to a range of informal entities. In Paragraph 3.3 we consider the question why 23 is a natural number (assuming it is).

In order to strengthen the language used to discuss "natural numberhood" I will make use of the following predicates and abbreviations for these:

- NN(x): x is a natural number.
- RNN(x): x is a representation of a natural number.
- PRNN(x): x is a preferred representation of a natural number.

I will assume that being comfortable with NN(23) and similar assertions is a necessity for any teacher of elementary mathematics. Each explanation of the ontology of natural numbers that must portray NN(23) as false (in essence) rather than as true creates practical problems because a question like: "which of the following natural numbers is larger, 37 of 389?" should not be rejected off hands simply on the ground that 37 or 389 cannot qualify as a natural number.

2 One abstract datatype, four datatypes

Following [6, 7] an abstract datatype may be understood as an isomorphism class of single-sorted or many sorted algebras. The instantiations of an abstract datatype, which are (concrete rather than abstract) datatypes. We will consider datatypes $\mathbb{N}_{ubdh}$ with normal forms in unary notation, $\mathbb{N}_{budh}$ with normal forms in binary notation, $\mathbb{N}_{dubh}$ with normal forms in decimal notation, $\mathbb{N}_{hubd}$ with normal forms in decimal notation. These structures are isomorphic, and

for that reason constitute realizations of the same abstract data type. In each case a datatype defining rewrite system is given.

2.1 Unary view

In Table 2 an algebraic specification is provided of the natural numbers with constants zero and one, and with successor, addition and multiplication. Successor terms, that is expressions involving zero and successor only serve as normal forms. We wil refer to the equations contained in this table as $E(\mathbb{N}_{ubdh})$.

Binary notation and decimal notation are explained by expanding terms into successor terms. This expansion involves a combinatorial explosion in size. That explosion renders the specification in Table 2 unfeasible as a term rewrite system from which an implementation can be generated.

Below we will consider adaptations of this specification, where normal forms are in binary notation and in decimal notation respectively. Specifications become far more lengthy and involved, but as a rewrite system the quality improves. We notice that when designing a complete term rewrite system to specify a given algebra a choice needs to be made regarding the normal forms. Obviously normal forms in unary notation, binary notation, and decimal notation are mutually exclusive.

In the unary view natural numbers are $0, S(0), S(S(0)), S(S(S(0))), ...$ Brackets are often left out, thus obtaining: $0, S0, SS0, SSS0, ...$ When confronted with $SSSS0$ (or SSSS0) a reader will recognize the expression as a natural number in successor notation, that is in unary view.

All other terms of the datatype are RNNs, though only the terms exclusively made from 0 and S are in classified as PRNN (and for that reason in NN).

2.2 Binary view

In Table 3 primitives for binary notation of natural numbers are specified by way of a complete term rewriting system. In the binary view natural numbers are identified with normal forms in binary notation. This datatype defining rewrite system produces the data type $\mathbb{N}_{bud}$. We refer to the equations contained in this table as $E(\mathbb{N}_{budh})$.

The specification defines a datatype which is isomorphic to the datatype defined by the specification in Table 2. In other words both datatypes belong to the same abstract datatype.

2.3 Decimal view

In Table 4 and Table 5 a CDDRS for a decimal view of natural numbers is displayed. For a decimal view one intends to have decimal normal forms. We refer to the equations contained in this table as $E(\mathbb{N}_{dubh})$.

$$0' \equiv 1$$
$$1' \equiv 2$$
$$2' \equiv 3$$
$$3' \equiv 4$$
$$4' \equiv 5$$
$$5' \equiv 6$$
$$6' \equiv 7$$
$$8' \equiv 9$$
$$9' \equiv A$$
$$A' \equiv B$$
$$B' \equiv C$$
$$C' \equiv D$$
$$D' \equiv E$$
$$E' \equiv F$$

Table 1: Enumeration of decimal/hexadecimal digits

$$x + 0 = x \tag{1}$$
$$x + S(y) = S(x + y) \tag{2}$$
$$x \cdot 0 = 0 \tag{3}$$
$$x \cdot S(y) = (x \cdot y) + x \tag{4}$$
$$\bigwedge_{i=0}^{i=E} [i' = S(i)] \tag{5}$$
$$x \ominus 0 = S(1) \cdot x \tag{6}$$
$$x \oslash 0 = S(9) \cdot x \tag{7}$$
$$x \otimes 0 = S(F) \cdot x \tag{8}$$
$$x \ominus 1 = S(x \ominus 0) \tag{9}$$
$$\bigwedge_{i=1}^{i=9} [(x \oslash i = (x \oslash 0) + i] \tag{10}$$
$$\bigwedge_{i=1}^{i=F} [(x \otimes i = (x \otimes 0) + i] \tag{11}$$

Table 2: Natural numbers in unary view ($\mathbb{N}_{ubdh}$)

$$0 \ominus 0 = 0 \tag{12}$$
$$0 \ominus 1 = 1 \tag{13}$$
$$S(0) = 1 \tag{14}$$
$$S(1) = 1 \ominus 0 \tag{15}$$
$$S(x \ominus 0) = x \ominus 1 \tag{16}$$
$$S(x \ominus 1) = S(x) \ominus 0 \tag{17}$$
$$x + 0 = x \tag{18}$$
$$0 + x = x \tag{19}$$
$$x + 1 = S(x) \tag{20}$$
$$1 + x = S(x) \tag{21}$$
$$(x \ominus 0) + (y \ominus 0) = (x + y) \ominus 0 \tag{22}$$
$$(x \ominus 0) + (y \ominus 1) = (x + y) \ominus 1 \tag{23}$$
$$(x \ominus 1) + (y \ominus 0) = (x + y) \ominus 1 \tag{24}$$
$$(x \ominus 1) + (y \ominus 1) = ((x + y) + 1) \ominus 0 \tag{25}$$
$$x \cdot 0 = 0 \tag{26}$$
$$0 \cdot x = 0 \tag{27}$$
$$x \cdot 1 = x \tag{28}$$
$$1 \cdot x = x \tag{29}$$
$$(x \ominus 0) \cdot y = (x \cdot y) \ominus 0 \tag{30}$$
$$x \cdot (y \ominus 0) = (x \cdot y) \ominus 0 \tag{31}$$
$$(x \ominus 1) \cdot y = (x \cdot y) \ominus 0 + y \tag{32}$$
$$x \cdot (y \ominus 1) = (x \cdot y) \ominus 0 + x \tag{33}$$
$$\bigwedge_{i=0}^{i=E} [i' = S(i)] \tag{34}$$
$$x \oslash 0 = S(9) \cdot x \tag{35}$$
$$x \otimes 0 = S(F) \cdot x \tag{36}$$
$$\bigwedge_{i=1}^{i=9} [x \oslash i = (x \oslash 0) + i] \tag{37}$$
$$\bigwedge_{i=1}^{i=F} [(x \otimes i = (x \otimes 0) + i] \tag{38}$$

Table 3: Natural numbers in binary view ($\mathbb{N}_{budh}$)

$$x + 0 = x \tag{39}$$

$$0 + x = x \tag{40}$$

$$\bigwedge_{i=1}^{i=8} [i + 1 = i'] \tag{41}$$

$$9 + 1 = 1 \oslash 0 \tag{42}$$

$$\bigwedge_{i=9}^{i=E} [i' = i + 1] \tag{43}$$

$$\bigwedge_{i=1}^{i=8} [x + i' = (x + i) + 1] \tag{44}$$

$$\bigwedge_{i=1}^{i=8} [i' + x = (x + i) + 1] \tag{45}$$

$$\bigwedge_{i=0}^{i=8} [(x \oslash i) + 1 = x \oslash i'] \tag{46}$$

$$\bigwedge_{i=0}^{i=8} [1 + (x \oslash i) = x \oslash i'] \tag{47}$$

$$(x \oslash 9) + 1 = (x + 1) \oslash 0 \tag{48}$$

$$1 + (x \oslash 9) = (x + 1) \oslash 0 \tag{49}$$

$$\bigwedge_{i=0}^{i=9} \bigwedge_{j=0}^{j=9} [(x \oslash i) + (y \oslash j) = (((x + y) \oslash i) + j] \tag{50}$$

$$\bigwedge_{i=0}^{i=9} [0 \oslash i = i] \tag{51}$$

$$S(x) = x + 1 \tag{52}$$

$$x \ominus 0 = S(1) \cdot x \tag{53}$$

$$x \ominus 1 = (x \ominus 0) + 1 \tag{54}$$

$$x \otimes 0 = S(F) \cdot x \tag{55}$$

$$\bigwedge_{i=1}^{i=F} [(x \otimes i = (x \otimes 0) + i] \tag{56}$$

Table 4: Natural numbers with addition in decimal view ($\mathbb{N}_{dubh}$ first part)

$$x \cdot 0 = 0 \tag{57}$$

$$0 \cdot x = 0 \tag{58}$$

$$\bigwedge_{i=0}^{i=8} [i' \cdot x = (i \cdot x) + x] \tag{59}$$

$$\bigwedge_{i=0}^{i=8} [x \cdot i' = (x \cdot i) + x] \tag{60}$$

$$\bigwedge_{i=0}^{i=9} [(x \oslash i) \cdot y = (x \cdot y) \oslash 0 + (i \cdot y)] \tag{61}$$

$$\bigwedge_{i=0}^{i=9} [x \cdot (y \oslash i) = (x \cdot y) \oslash 0 + (x \cdot i)] \tag{62}$$

Table 5: Multiplication in decimal view ($\mathbb{N}_{dubh}$ continued)

2.4 hexadecimal view

In Table 6 and Table 7 a CDDRS for a hexadecimal view of natural numbers is displayed. We refer to the equations contained in both tables as $E(\mathbb{N}_{hubd})$.

2.5 Assessment of the four rewrite systems

We notice that the CDDRS's provide equivalent algebraic specifications, that is $I(\Sigma_{\mathbb{N}}, E(\mathbb{N}_{ubdh}) \simeq I(\Sigma_{\mathbb{N}}, E(\mathbb{N}_{budh}) \simeq I(\Sigma_{\mathbb{N}}, E(\mathbb{N}_{dubh}) \simeq I(\Sigma_{\mathbb{N}}, E(\mathbb{N}_{hubd})$. Once a choice for the format of normal forms has been made four criteria remain for the design of a CDDRS:

1. readability (an informal notion)
2. conciseness, (either measured in terms of the number of rules or of the sum of their sizes),
3. effectiveness (how fast can normal forms be found in terms of numbers of steps, given a know reduction strategy),
4. number of auxiliary functions (auxiliary functions may be needed when encoding fast algorithms for multiplication in a CDDRS).

The unary view is readable, concise, and does without auxiliary functions. Its main deficiency is that it algorithmic content is problematic in that, for inputs in binary notation or in decimal notation normalization of terms of the form $t + r$ takes a number of steps growing exponentially in the size of t and r. The other two specifications don't have an algorithmic deficiency though each is quite specific for its own notation requiring conversions to and

$$x + 0 = x \tag{63}$$

$$0 + x = x \tag{64}$$

$$\bigwedge_{i=1}^{i=E} [i + 1 = i'] \tag{65}$$

$$F + 1 = 1 \otimes 0 \tag{66}$$

$$\bigwedge_{i=1}^{i=E} [x + i' = (x + i) + 1] \tag{67}$$

$$\bigwedge_{i=1}^{i=E} [i' + x = (x + i) + 1] \tag{68}$$

$$\bigwedge_{i=0}^{i=E} [(x \otimes i) + 1 = x \otimes i'] \tag{69}$$

$$\bigwedge_{i=0}^{i=E} [1 + (x \otimes i) = x \otimes i'] \tag{70}$$

$$(x \otimes F) + 1 = (x + 1) \otimes 0 \tag{71}$$

$$1 + (x \otimes F) = (x + 1) \otimes 0 \tag{72}$$

$$\bigwedge_{i=0}^{i=F} \bigwedge_{j=0}^{j=F} [(x \otimes i) + (y \otimes j) = (((x + y) \otimes i) + j] \tag{73}$$

$$\bigwedge_{i=0}^{i=F} [0 \otimes i = i] \tag{74}$$

$$S(x) = x + 1 \tag{75}$$

$$x \ominus 0 = S(1) \cdot x \tag{76}$$

$$x \ominus 1 = (x \ominus 0) + 1 \tag{77}$$

$$x \oslash 0 = S(9) \cdot x \tag{78}$$

$$\bigwedge_{i=1}^{i=9} [(x \oslash i = (x \oslash 0) + i] \tag{79}$$

Table 6: Natural numbers with addition in hexadecimal view ($\mathbb{N}_{hdub}$ first part)

$$x \cdot 0 = 0 \tag{80}$$

$$0 \cdot x = 0 \tag{81}$$

$$\bigwedge_{i=0}^{i=F} [i' \cdot x = (i \cdot x) + x] \tag{82}$$

$$\bigwedge_{i=0}^{i=F} [x \cdot i' = (x \cdot i) + x] \tag{83}$$

$$\bigwedge_{i=0}^{i=F} [(x \otimes i) \cdot y = (x \cdot y) \otimes 0 + (i \cdot y)] \tag{84}$$

$$\bigwedge_{i=0}^{i=F} [x \cdot (y \otimes i) = (x \cdot y) \otimes 0 + (x \cdot i)] \tag{85}$$

Table 7: Multiplication in hexadecimal view ($\mathbb{N}_{hdub}$ continued)

from other notations before and after each operation. Conciseness and readability are less prominent with the CDDRSs for $\mathbb{N}_{budh}$, $\mathbb{N}_{dubh}$, and $\mathbb{N}_{hubd}$.

3 What are natural numbers? A plurality of datatypes

At this stage one may wish to understand natural numbers as the elements of the carrier of an abstract data type. Unfortunately, abstract datatypes being isomorphism classes have no elements. Only carriers of data types, that is carriers of instances of abstract data types, have elements.

That is, we know of the structure of natural numbers as an abstract datatype $I(\Sigma_{\mathbb{N}}, E(\mathbb{N}_{budh}))$, while individual numbers are only observed as values in carriers of data types. For the three dataypes at hand these values are different closed terms over the common signature $\Sigma_{\mathbb{N}}$.

For each base of a number system a rewrite system can be designed having normal forms characteristic of that base. However, for a base b different from 1, 2, 10, and 16 an extension of the syntax is needed if a base b view on natural numbers is to be specified.

3.1 Notations for natural numbers

The proposal of this paper provides 8 notational schemes, conventions, or formats, for natural numbers: unary, binary, decimal, and hexadecimal format with and without brackets and operator symbols for appending a digit.

These representations all share the virtue that for each individual scheme numbers have unique representation, which we take for a criterion that allows one to say that such a representation (expression for a natural number in a particular format) is a natural number, rather

than that it merely represents one.

Nevertheless one may insist that say 2 is no more than a representation of a natural number. In the light of our abstract datatype, and using only the bracketed expressions that number is a four-tuple: $(S(S(0)), 1 \ominus 0, 2, 2)$. In other words taking our abstract datatype for the natural numbers as a point of departure numbers are quadruples of terms and at first inspection the members of these quadruples are merely representations of numbers or in other words notations of numbers, and such elements do not constitute the numbers themselves.

3.2 Taking the notational variation seriously

If one insists to take unbracketed expressions into account as well a number (say 2) becomes an 8-tuple: $(S(S(0)), SS0, 1 \ominus 0, 10, 2, 2, 2, 2)$. At this stage the ordering in the tuple matters because it helps to avoid confusion between the various logarithmic notations. Needless to say that when writing a number in practice one prefers to provide only one of the members of the 8-tuple. Moreover one often prefers unbracketed decimal notation.

3.2.1 Advantages of the multiple datatype understanding of natural numbers

Understanding natural numbers as tuples of entities in a range (in this case four) of datatypes is reasonable and complies with ordinary practice. Here are some further comments about the practicality of that proposal.

- The signature $\Sigma_{\mathbb{N}}$ is a rather natural choice which unifies three notations for natural numbers: unary, binary, and decimal.
- For each notation unique normal forms provide canonical representations of natural numbers. One may indeed say that 5 is the first (leading, decimal) digit of 578 because only 578 is a normal form representing that number.
- We may state that 578 is a natural number although from a principled point of view it is merely a representation of a natural number. We notice:
  - This use of the language is unproblematic because in its kind the representation is unique and the kind (decimal notation) can be derived from the representation.
  - That 578 abbreviates $(5 \oslash 7) \oslash 8$ creates no confusion, as the introduction of these brackets and operator symbols follows in a unique and deterministic manner.
- By having multiple views different forms of algorithms for functions and of methods to define functions can be supported.

3.2.2 Disadvantages of a multiple datatype understanding of natural numbers

The multiple datatype understanding of natural numbers has at least these weaknesses.

- The signature $\Sigma_{\mathbb{N}}$ is a rather arbitrary choice, both smaller and larger signatures will be useful for the same purpose of identifying natural numbers. There is no "natural" signature for the natural numbers.
- We provide only four datatypes for the abstract datatype at hand while infinitely many such datatypes might relevant.
- In our setup a natural number, say 57, is identified with four different terms (each of those bringing with it its own simplified form) over the same signature which have the same meaning in the abstract datatype specified by three different specifications sharing the same signature. This is plurality of terms denoting the same entity demonstrates merely an approximation of the level of abstraction provided by an abstract datatype.

3.3 Why 23 is a natural number, and why 023 isn't?

The point of departure for a pragmatic philosophy of natural numbers based on a CDDRS portfolio is my assumption that (the teaching of) elementary mathematics needs to make use of straightforward terminology while still admitting some precise analysis. I will consider in detail the text fragment "23" and consider the question why 23 might qualify as a natural number.

Following the definitions in Paragraph 1.5, NN(23) $\equiv$ "23 is a natural number". About NN(23) I maintain the following viewpoints:

1. In school teaching one needs to be able to make assertions like these:
   - NN(23),
   - "23 is a prime number" (and therefore NN(23)).
   - "-23 is an integer",
   - "$23 + (-23) = 0$, that's how addition works for natural numbers and their negative versions"
2. It is of no use in a school setting if one can only say: "23 is a natural number in decimal notation".
3. If one states that "23 is a notation for a natural number", the obvious reply is to ask: "for which natural number". And then the answer can't be 23 unless 23 is a natural number (and not merely a notation for a natural number).
4. Notice that 23 is a different natural number in hexadecimal notation (in fact 35), although it is also the 22th successor of 1 in hexadecimal notation.
5. There is a sliding scale:
   - we may want to say: "$\sum_{n=1}^{n=\infty} \frac{1}{n^2}$ is a power series converging to a real number" rather than "$\sum_{n=1}^{n=\infty} \frac{1}{n^2}$ is a a real number".
   - But we will not deny that "$\sum_{n=1}^{n=\infty} \frac{1}{n^2}$ is a real number" can be correctly asserted.

- If we hold that integers are defined as pairs of naturals (say writing s for shift, $s(a, b)$ for $a, b \in \mathbb{N}$ (thus representing $a + (-b)$) then -23 is merely a notation for $s(0, 23)$.[1]
- If we write BV("P = NP") for "P = NP is a Boolean value", many readers may object to a claim of validity for the assertion BV("P = NP") on the grounds that it is unclear which value is meant (that's the famous open problem), and because "P = NP" is at best an indirect specification of a Boolean value hardly deserving the description "expression for a Boolean value", let alone the simplified "Boolean value".

6. The assertion NN(23) may be contrasted with NN(023). I hold that it is not plausible to assert both NN(23) and NN(0023) at the same time. In a setting where both 23 and 023 occur as expressions denoting natural numbers, it is reasonable to say that 23 is a decimal representation (of the number it denotes) without leading zeroes and that 023 is a representation with a leading zero, while 23 is a simplified form of 023, with simplification involving the removal of leading zeroes.

   It is implausible to say that 23 is a natural number without leading zeroes, and that 023 is a natural number with leading zeroes. Natural numbers don't have that attribute, instead the presence or absence of leading zeroes is a property of some logarithmic number representations.

7. One may formalize the above argument about 23 and 023. Formalization leads to the following implication: "if NN(23) then ¬ NN(023)". By consequence "if NN(023) then ¬ NN(23)".

   In order to assert NN(23) one implicitly assumes notational conventions that exclude 023. If such conventions are absent than 23 advances no further than to the status of a representation of a natural number (RNN(23) in the notation of Paragraph 1.5).

   Now one finds: "if (RNN(23) and RNN(023)) then ¬ NN(23)". Here it is implicitly assumed that the condition RNN(23) and RNN(023) is evaluated in the same context. Being equally true escapes from two valued logic, and this asks for some rephrasing.

   One may advance further and write PRNN(w) for "w is a preferred representation of a natural number". Clearly PRNN(w) implies RNN(w). Now PRNN(23) may be assumed to imply imply NN(23) but a less demanding interpretation is possible. In any case one finds: "if (RNN(23) and ¬ PRNN(23)) then ¬ NN(23)".

8. One may contemplate whether or not 23 is a natural number value (instead of being a natural number) and 22+1 is also a natural number, (but not a natural number value), and if "23 = 22+1" (assuming its truth) rests on the assumption that 23 and 22+1 are the same kind of entity. We will hold that:

   - 23 is a natural number and it is a natural number value as well.

[1] This viewpoint would be criticized by those who feel that the foundational approach to mathematical structures which induces the construction of integers as pairs of naturals is not meant to have this pedantic influence on the use of "ordinary" mathematical language.

- 22+1 is a natural number because it denotes a natural number.
- 22+1 is a natural number and it is not a natural number value, though the value of 22+1 is a natural number.
- 22+1 is a natural number form (NNF, form is assumed to be equivalent with expression).
- $\alpha \equiv 3^2+3^2+2^2+1$ is a representation of 23 as a sum of four squares. This assertion can be valid only if the actual form of an NNF matters and can have properties that are not universal for NNs.
- $\beta \equiv 4^2+2^2+1+1+1$ is a representation of 23 as a sum of five squares. Now α and β are different representations (if a natural number as a sum of squares).
- $\alpha \not\equiv \beta$ in a realistic sense. However in that same sense (as representations in terms of sums of squares): $4^2+2^2+1+1+1 \equiv 4^2+2^2+1^2+1^2+1^2 \equiv 1^2+1^2+1^2+4^2+2^2$.
- A difference between two NNFs is determined (accepted, found, agreed) in the context of an objective of representational form. It is plausible that differences between representational forms are defined within dedicated subclasses of NNF. In other words by merely writing that $4^2+2^2+1+1+1 \not\equiv 2\cdot 10+3$ we fail to indicate in what quality (property) both expressions differ.

9. If we write $\phi \equiv 23+23=46$, ϕ will be considered a valid assertion by most readers. In ϕ we find two different occurrences of 23 which are at the same time equal, in the sense that both are natural numbers which, moreover, can only be the same natural numbers. It appears that we have an inconsistency.

   In order to explain this case I propose that as soon as one starts discussing occurrences of 23 one needs to acknowledge that NN(23) is meant modulo "an arbitrary occurrence of". Thus NN(23) can be rephrased as: each occurrence of 23 is an occurrence of a natural number. In other words, the meaning of NN(23) depends on the abstraction level at which one is considering the matter. At a lower level of abstraction a different definition of NN(23) may be needed.

10. Of course one may introduce an equivalence $\equiv$ on decimal notations with and without leading zeroes so that $23 \equiv 023$, and subsequently work modulo that equivalence. However, doing so creates a situation in which the assertion that removing leading zeroes turns 023 into 23 is wrong because both are equal modulo $\equiv$ and there is no such thing as leading zeroes modulo this equivalence.

4 Concluding remarks

We have proposed four datatype defining rewrite systems for as many different views on the natural numbers. These views are instances of the same abstract datatype for which each of the rewrite systems constitutes an equational initial algebra specification at the same time.

By considering a limited plurality of datatypes instantiating the same underlying abstract datatype we determine an intermediate abstractions for the individual natural numbers, lying in between the number as an entity in a specific datatype and the abstraction of that entity in an abstract datatype.

References

[1] J. A. Bergstra, I. Bethke, and P.H. Rodenburg. A propositional logic with 4 values: true, false, divergent, and meaningless. *J. of Applied Non-Classical Logics*, 5(2): 199–218 (1995).

[2] J.A. Bergstra and C.A. Middelburg. Thread algebra for strategic interleaving. *Formal Aspects of Computing*, 19(4):445–474 (2007).

[3] J.A. Bergstra and C.A. Middelburg. Inversive meadows and divisive meadows. *Journal of Applied Logic*, 9(3): 203–220 (2011).

[4] J.A. Bergstra and A. Ponse. Proposition algebra. *ACM Transactions on Computational Logic*, 12 (3), Article 21 (36 pages), 2011.

[5] Jan A. Bergstra, Alban Ponse, and Daan J.C. Staudt. Short-circuit logic. `arXiv.org:1010.3674v4` (2013).

[6] Jan A. Bergstra, and John V. Tucker. Algebraic specifications of computable and semi-computable data types. *Theoretical Computer Science*, 50 (2), pp. 137–181, (1987).

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

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

[9] Manfred Broy, Martin Wirsing, and Claude Pair. A systematic study of models of abstract data types. *Theoretical Computer Science,* 33 (2), 139-174 (1984).

[10] Marie-Claude Gaudel and Perry R. James. Testing algebraic data types and processes: a unifying theory. *Formal Aspects of Computing,* 10 (5-6) pp. 436-451, (1998).

[11] Anonymous Wikipedian(s). Hexadecimal, `http://en.wikipedia.org/wiki/hexadecimal` (2014).

[12] Martin Wirsing, Algebraic Specification. *in: Handbook of theoretical computer science, (vol. B), MIT Press,* pp. 675-788 (1991).

Electronic Reports Series of section Theory of Computer Science

Within this series the following reports appeared.

[TCS1408] J.A. Bergstra, *Bitcoin: Informational Money en het Einde van Gewoon Geld*, section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1407] J.A. Bergstra, *Four Complete Datatype Defining Rewrite Systems for an Abstract Datatype of Natural Numbers*, section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1406v2] J.A. Bergstra, *Bitcoin and Islamic Finance (version 2)*, section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1406] J.A. Bergstra, *Bitcoin and Islamic Finance*, section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1405] J.A. Bergstra, *Rekenen in een Conservatieve Schrapwet Weide*, section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1404] J.A. Bergstra, *Division by Zero and Abstract Data Types*, section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1403] J.A. Bergstra, I. Bethke, and A. Ponse, *Equations for Formally Real Meadows*, section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1402] J.A. Bergstra and W.P. Weijland, *Bitcoin, a Money-like Informational Commodity*, section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1401] J.A. Bergstra, *Bitcoin, een "money-like informational commodity"*, section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1301] B. Diertens, *The Refined Function-Behaviour-Structure Framework*, section Theory of Computer Science - University of Amsterdam, 2013.

[TCS1202] B. Diertens, *From Functions to Object-Orientation by Abstraction*, section Theory of Computer Science - University of Amsterdam, 2012.

[TCS1201] B. Diertens, *Concurrent Models for Object Execution*, section Theory of Computer Science - University of Amsterdam, 2012.

[TCS1102] B. Diertens, *Communicating Concurrent Functions*, section Theory of Computer Science - University of Amsterdam, 2011.

[TCS1101] B. Diertens, *Concurrent Models for Function Execution*, section Theory of Computer Science - University of Amsterdam, 2011.

[TCS1001] B. Diertens, *On Object-Orientation*, section Theory of Computer Science - University of Amsterdam, 2010.

Within former series (PRG) the following reports appeared.

[PRG0914] J.A. Bergstra and C.A. Middelburg, *Autosolvability of Halting Problem Instances for Instruction Sequences*, Programming Research Group - University of Amsterdam, 2009.

[PRG0913] J.A. Bergstra and C.A. Middelburg, *Functional Units for Natural Numbers*, Programming Research Group - University of Amsterdam, 2009.

[PRG0912] J.A. Bergstra and C.A. Middelburg, *Instruction Sequence Processing Operators*, Programming Research Group - University of Amsterdam, 2009.

[PRG0911] J.A. Bergstra and C.A. Middelburg, *Partial Komori Fields and Imperative Komori Fields*, Programming Research Group - University of Amsterdam, 2009.

[PRG0910] J.A. Bergstra and C.A. Middelburg, *Indirect Jumps Improve Instruction Sequence Performance*, Programming Research Group - University of Amsterdam, 2009.

[PRG0909] J.A. Bergstra and C.A. Middelburg, *Arithmetical Meadows*, Programming Research Group - University of Amsterdam, 2009.

[PRG0908] B. Diertens, *Software Engineering with Process Algebra: Modelling Client / Server Architecures*, Programming Research Group - University of Amsterdam, 2009.

[PRG0907] J.A. Bergstra and C.A. Middelburg, *Inversive Meadows and Divisive Meadows*, Programming Research Group - University of Amsterdam, 2009.

[PRG0906] J.A. Bergstra and C.A. Middelburg, *Instruction Sequence Notations with Probabilistic Instructions*, Programming Research Group - University of Amsterdam, 2009.

[PRG0905] J.A. Bergstra and C.A. Middelburg, *A Protocol for Instruction Stream Processing*, Programming Research Group - University of Amsterdam, 2009.

[PRG0904] J.A. Bergstra and C.A. Middelburg, *A Process Calculus with Finitary Comprehended Terms*, Programming Research Group - University of Amsterdam, 2009.

[PRG0903] J.A. Bergstra and C.A. Middelburg, *Transmission Protocols for Instruction Streams*, Programming Research Group - University of Amsterdam, 2009.

[PRG0902] J.A. Bergstra and C.A. Middelburg, *Meadow Enriched ACP Process Algebras*, Programming Research Group - University of Amsterdam, 2009.

[PRG0901] J.A. Bergstra and C.A. Middelburg, *Timed Tuplix Calculus and the Wesseling and van den Berg Equation*, Programming Research Group - University of Amsterdam, 2009.

[PRG0814] J.A. Bergstra and C.A. Middelburg, *Instruction Sequences for the Production of Processes*, Programming Research Group - University of Amsterdam, 2008.

[PRG0813] J.A. Bergstra and C.A. Middelburg, *On the Expressiveness of Single-Pass Instruction Sequences*, Programming Research Group - University of Amsterdam, 2008.

[PRG0812] J.A. Bergstra and C.A. Middelburg, *Instruction Sequences and Non-uniform Complexity Theory*, Programming Research Group - University of Amsterdam, 2008.

[PRG0811] D. Staudt, *A Case Study in Software Engineering with PSF: A Domotics Application*, Programming Research Group - University of Amsterdam, 2008.

[PRG0810] J.A. Bergstra and C.A. Middelburg, *Thread Algebra for Poly-Threading*, Programming Research Group - University of Amsterdam, 2008.

[PRG0809] J.A. Bergstra and C.A. Middelburg, *Data Linkage Dynamics with Shedding*, Programming Research Group - University of Amsterdam, 2008.

[PRG0808] B. Diertens, *A Process Algebra Software Engineering Environment*, Programming Research Group - University of Amsterdam, 2008.

[PRG0807] J.A. Bergstra, S. Nolst Trenite, and M.B. van der Zwaag, *Tuplix Calculus Specifications of Financial Transfer Networks*, Programming Research Group - University of Amsterdam, 2008.

[PRG0806] J.A. Bergstra and C.A. Middelburg, *Data Linkage Algebra, Data Linkage Dynamics, and Priority Rewriting*, Programming Research Group - University of Amsterdam, 2008.

[PRG0805] J.A. Bergstra, S. Nolst Trenite, and M.B. van der Zwaag, *UvA Budget Allocatie Model*, Programming Research Group - University of Amsterdam, 2008.

[PRG0804] J.A. Bergstra and C.A. Middelburg, *Thread Algebra for Sequential Poly-Threading*, Programming Research Group - University of Amsterdam, 2008.

[PRG0803] J.A. Bergstra and C.A. Middelburg, *Thread Extraction for Polyadic Instruction Sequences*, Programming Research Group - University of Amsterdam, 2008.

[PRG0802] A. Barros and T. Hou, *A Constructive Version of AIP Revisited*, Programming Research Group - University of Amsterdam, 2008.

[PRG0801] J.A. Bergstra and C.A. Middelburg, *Programming an Interpreter Using Molecular Dynamics*, Programming Research Group - University of Amsterdam, 2008.

The above reports and more are available through the website: www.science.uva.nl/research/prog/

Electronic Report Series

section Theory of Computer Science
Faculty of Science
University of Amsterdam

Science Park 904
1098 XG Amsterdam
the Netherlands

www.science.uva.nl/research/prog/