University of Amsterdam
Theory of Computer Science

Three Datatype Defining Rewrite Systems for Datatypes of Integers each extending a Datatype of Naturals (version 2)

J.A. Bergstra
A. Ponse

Report TCS1409v2 August 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

A. Ponse

section Theory of Computer Science
Faculty of Science
University of Amsterdam

Science Park 904
1098 XH Amsterdam
the Netherlands

tel. +31 20 525.7592
e-mail: A.Ponse@uva.nl

Theory of Computer Science Electronic Report Series

Three Datatype Defining Rewrite Systems for Datatypes of Integers each extending a Datatype of Naturals*

Jan A. Bergstra and Alban Ponse
Informatics Institute, University of Amsterdam
j.a.bergstra@uva.nl a.ponse@uva.nl

August 21, 2014

Abstract

Integer arithmetic is specified according to three views: unary, binary, and decimal notation. In each case we find a ground confluent and terminating datatype defining rewrite system. In each case the resulting datatype is a canonical term algebra which extends a corresponding canonical term algebra for natural numbers. For each view, we also consider an alternative rewrite system.

Keywords and phrases: Equational specification, initial algebra, datatype defining rewriting system, abstract datatype.

Contents

1 Introduction **2**
1.1 Digits and rewrite rules in equational form . . . 2
1.2 A signature for integers . . . 3

2 One ADT, three datatypes **4**
2.1 Unary view . . . 4
2.2 Binary view . . . 7
2.3 Decimal view . . . 9

3 Alternative DDRS's for integers with digit tree constructors **11**
3.1 Unary view with digit tree constructor . . . 11
3.2 Binary view with digit tree constructor . . . 13
3.3 Decimal view with digit tree constructor . . . 14

4 Concluding remarks **15**

References **17**

*Version 2: other notation for append digit functions; better lay-out for rule schemata; Section 2.1 extended with unary append zero function; new Section 3, in which we briefly discuss some alternative DDRS's.

$0' \equiv 1$	$3' \equiv 4$	$6' \equiv 7$
$1' \equiv 2$	$4' \equiv 5$	$7' \equiv 8$
$2' \equiv 3$	$5' \equiv 6$	$8' \equiv 9$

Figure 1: Enumeration and successor notation of digits of type $\mathbb{Z}$

1 Introduction

Using the specifications for natural numbers from [1] we develop specifications for datatypes of integers. We will entertain the strategy of [1] to develop different views characteristic for unary notation, binary notation, and decimal notation respectively. Each of the specifications is a so-called DDRS (datatype defining rewrite system) and consists of a number of equations that define a term rewriting system by orienting the equations from left-to-right. A DDRS must be strongly terminating and ground confluent.

This paper is a sequel to the report [1] which deals with DDRS's for the natural numbers and it constitutes a further stage in the development of a family of arithmetical datatypes with corresponding specifications. The resulting specifications (DDRS's) incorporate different "views" on the same abstract datatype. 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 in [1] with respect to conventional notations in such a way that syntactic confusion between these notations cannot arise. In this paper, the *hexadecimal view* is left out as that seems to be an unusual viewpoint for integer arithmetic.

It seems to be the case that for the unary view the specification of the integers (given in Table 2) is entirely adequate, whereas all subsequent specifications for binary view and decimal view may provide no more than a formalization of a topic which must be somehow understood before taking notice of that same formalization. It remains to be seen to what extent the first DDRS for the unary case may serve exactly that expository purpose.

The strategy of the work is somewhat complicated: on the one hand we look for specifications that may genuinely be considered introductory, that is, descriptions that can be used to construct the datatype at hand for the first time in the mind of a person. On the other hand awareness of the datatype in focus may be needed to produce an assessment of the degree of success achieved in the direction of the first objective.

1.1 Digits and rewrite rules in equational form

Digits are $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, and are ordered in the common way:

$$0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9.$$

For the digits $0, 1, 2, 3, 4, 5, 6, 7, 8$ we denote with i' the successor digit of i in the given enumeration. In Figure 1 the successor notation on digits is specified as a transformation of syntax, and we adopt this notation throughout the paper.

We will list rewrite rules in the form of equations $t = r$ to be interpreted from left-to-right, and we will add tags of the form

$$[\mathrm{Nn}] \quad t = r$$

for reference, with "N" some name and "n" a natural number (in ordinary, decimal notation). Furthermore, for $k, \ell \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $k < \ell$, the notation

$$[\mathrm{Nn}.i]_{i=k}^{\ell} \quad t = r$$

represents the set of $\ell - k + 1$ rewrite rules

$$\{\ [\mathrm{Nn}.k] \quad t[k/i] = r[k/i], \quad \ldots \quad , [\mathrm{Nn}.\ell] \quad t[\ell/i] = r[\ell/i]\ \},$$

thus with i instantiated from k to ℓ. In the paper [14] of Walters and Zantema, such notational devices are called *rule schemata*, and occasionally, we will use these with two "digit counters", as in

$$[\mathrm{Nn}.i.j]_{i,j=k}^{\ell} \quad t = r.$$

1.2 A signature for integers

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

1. A sort $\mathbb{Z}$,
2. For digits the ten constants $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$,
3. Three one-place functions $S, P, - : \mathbb{Z} \to \mathbb{Z}$, "successor", "predecessor", and "minus", respectively,
4. Addition and multiplication (infix) $+, \cdot : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$,
5. Two one-place functions (postfix) $_ :_b 0, _ :_b 1 : \mathbb{Z} \to \mathbb{Z}$, "binary append zero" and "binary append one", these functions will be used for binary notation,
6. Ten one-place functions (postfix)

   $$_ :_d 0, _ :_d 1, _ :_d 2, _ :_d 3, _ :_d 4, _ :_d 5, _ :_d 6, _ :_d 7, _ :_d 8, _ :_d 9 : \mathbb{Z} \to \mathbb{Z},$$

   "decimal append zero", ...,"decimal append nine", to be used for decimal notation.

The "append <digit name>" functions defined in items 5 and 6 can be viewed as instantiations of more general two-place "append" functions, but that would require the introduction of sorts for bits (binary digits) and for decimal digits. However, we prefer to keep the signature single-sorted and that is why we instantiate such "digit append" functions per digit to unary functions and why we use postfix notation for applications of these functions. E.g.,

$$(9 :_d 7) :_d 5 \quad \text{and} \quad ((1 :_b 0) :_b 0) :_b 1$$

represent the decimal number 975, and the binary number 1001, respectively.

For the unary view the normal forms are the classical successor terms, that is

$$0, S(0), S(S(0)), \ldots,$$

and all minus instances $-(t)$ for each such non-zero normal form t, e.g. $-(S(S(0)))$. If no confusion can arise, we abbreviate $-(t)$ to $-t$, as in $-x$. As an alternative (and introduction to subsequent DDRS's) we shall briefly consider a DDRS that is based on a "unary append zero" function.

For the binary view and for the decimal view, we provide one DDRS for each. Normal forms are all appropriate digits, all applications of the respective append functions to a non-zero normal form, and all minus instances $-t$ for each such normal form t that differs from 0. Thus $-(((1 :_b 0) :_b 0) :_b 1)$ is an example of a normal form in binary view, and $-((9 :_d 7) :_d 5)$ is one in decimal view.

[S1]	$x + 0 = x$	$[\mathrm{S5}.i]_{i=0}^{8}$	$i' = S(i)$
[S2]	$x + S(y) = S(x + y)$	$[\mathrm{S6}.i]_{i=0}^{1}$	$x :_b i = (x \cdot S(1)) + i$
[S3]	$x \cdot 0 = 0$	$[\mathrm{S7}.i]_{i=0}^{9}$	$x :_d i = (x \cdot S(9)) + i$
[S4]	$x \cdot S(y) = (x \cdot y) + x$		

Table 1: $\mathbb{N}_{ubd}$, natural numbers in unary view

2 One ADT, three datatypes

An abstract datatype (ADT) may be understood as the isomorphism class of its instantiations which are datatypes. The datatypes considered in [1] are so-called canonical term algebras which means that carriers are non-empty sets of closed terms which are closed under taking subterms.

2.1 Unary view

Table 1 provides a DDRS for the natural numbers and defines the datatype $\mathbb{N}_{ubd}$. Minus and predecessor are absent in this datatype. Successor terms, that is expressions involving zero and successor only, serve as normal forms for the datatype $\mathbb{N}_{ubd}$. This DDRS contains the well-known rules [S1] – [S4] and the rules $[\mathrm{S5}.i]_{i=0}^{8}$ – $[\mathrm{S7}.i]_{i=0}^{9}$ (21 rules) that serve the rewriting of binary and decimal notation.

In Table 2 a DDRS is provided of the integer numbers $\mathbb{Z}_{ubd}$ with successor, predecessor, addition, and multiplication, which are defined by the rules [u1] – [u14]. We notice that we do not need equations for rewriting

$$(-x) \cdot y$$

because multiplication is defined by recursion on its right-argument, and that is why equation [u14] is sufficient, and why addition is defined by recursion on both its arguments and also requires [u11]. Like before, the 21 rules $[\mathrm{u15}.i]_{i=0}^{8}$ – $[\mathrm{u17}.i]_{i=0}^{9}$ serve the rewriting of binary and decimal notation.

In Table 3 one finds a listing of equations that are true in the datatype $\mathbb{Z}_{ubd}$ that is specified by the DDRS of Table 2. This ensures that these rewrite rules (viewed as equations) are semantic consequences of the equations for commutative rings.

So, binary and decimal notation are defined by expanding terms into successor terms. This expansion involves a combinatorial explosion in size and renders the specification in Tables 1 and 2 irrelevant as term rewrite systems from which an efficient implementation can be generated. In the sequel we will consider specifications where normal forms are in binary notation and in decimal notation, respectively, that also employ the successor and predecessor functions. These specifications are far more lengthy and involved, but as DDRS's their quality improves because normal forms are smaller and are reached in fewer rewriting steps.

In order to give a smooth introduction to the subsequent DDRS's (for binary view and decimal view), we end with a brief exposition of a very simple alternative to the above DDRS's. Consider the extension of the signature $\Sigma_{\mathbb{Z}}$ defined in Section 1.2 with a one-place function (postfix)

$$_ :_u : \mathbb{Z} \to \mathbb{Z},$$

the "unary append zero" function, or briefly: *zero append*, which can be used as an alternative for unary notation.

[u1]	$-0 = 0$	[u15.i]$_{i=0}^{8}$	$i' = S(i)$
[u2]	$-(-x) = x$	[u16.i]$_{i=0}^{1}$	$x :_b i = (x \cdot S(1)) + i$
[u3]	$P(0) = -S(0)$	[u17.i]$_{i=0}^{9}$	$x :_d i = (x \cdot S(9)) + i$
[u4]	$P(S(x)) = x$		
[u5]	$P(-x) = -S(x)$		
[u6]	$S(-(S(x))) = -x$		
[u7]	$x + 0 = x$		
[u8]	$0 + x = 0$		
[u9]	$x + S(y) = S(x + y)$		
[u10]	$S(x) + y = S(x + y)$		
[u11]	$(-x) + (-y) = -(x + y)$		
[u12]	$x \cdot 0 = 0$		
[u13]	$x \cdot S(y) = (x \cdot y) + x$		
[u14]	$x \cdot (-y) = -(x \cdot y)$		

Table 2: $\mathbb{Z}_{ubd}$, integer numbers in unary view

$x + (y + z) = (x + y) + z$		(1)
$x + y = y + x$		(2)
$x + 0 = x$		(3)
$x + (-x) = 0$		(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)
$S(x) = x + 1$		(9)
$P(x) = x + (-1)$		(10)
$x :_b i = (x + x) + i$	for $i \in \{0, 1\}$	(11)
$x :_d i = (\underline{10} \cdot x) + \underline{i}$	for $i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $\underline{0} = 0,\ \underline{i'} = \underline{i} + 1,\ \underline{10} = \underline{9} + 1$	(12)

Table 3: Equations valid in $\mathbb{Z}_{ubd}$, where (1) − (8) axiomatize commutative rings

[u′1]	$x + 0 = x$
[u′2]	$x + (y :_u 0) = (x :_u 0) + y$
[u′3]	$x \cdot 0 = 0$
[u′4]	$x \cdot (y :_u 0) = (x \cdot y) + x$

Table 4: $\mathbb{N}_{u'}$, natural numbers in unary view with zero append

Normal forms in this alternative unary view are 0 for zero, and applications of the zero append function that define all successor values (each natural number n is represented by n applications of the zero append and can be seen as representing a sequence of 0's of length $n + 1$). Furthermore, all minus instances $-t$ for each such normal form t that differs from 0 define the negative integers, so

$$-((0 :_u 0) :_u 0)$$

is an example of a normal form in this unary view (and it represents -2 in decimal notation).

Addition and multiplication are easy to define: Table 4 provides a DDRS for the natural numbers $\mathbb{N}_{u'}$. Termination follows with the use of a weight function, and also ground confluence follows easily.

The DDRS that defines the extension of $\mathbb{N}_{u'}$ to integer numbers $\mathbb{Z}_{u'}$ is given in Table 5 below.

It is immediately clear how rules for rewriting to unary notation with successor and predecessor, and binary and decimal notation can be defined, but we refrain from doing so and stick to the signature $\Sigma_{\mathbb{Z}}$ defined in Section 1.2 because the main purpose of the specifications in Table 4 and Table 4+5 is to introduce working with the "append functions", as we will do in our definitions of the datatypes in the subsequent sections.

[u′5]	$-0 = 0$
[u′6]	$-(-x) = x$
[u′7]	$0 + x = x$
[u′8]	$(x :_u 0) + (-(y :_u 0)) = x + (-y)$
[u′9]	$(-(y :_u 0)) + (x :_u 0) = x + (-y)$
[u′10]	$(-x) + (-y) = -(x + y)$
[u′11]	$x \cdot (-y) = -(x \cdot y)$
[u′12]	$(-x) \cdot y = -(x \cdot y)$

Table 5: $\mathbb{Z}_{u'}$, integer numbers in unary view with zero append, continuation of Table 4

$[\text{b1}.i]_{i=0}^{1}$	$0 :_b i = i$
[b2]	$S(0) = 1$
[b3]	$S(1) = 1 :_b 0$
[b4]	$S(x :_b 0) = x :_b 1$
[b5]	$S(x :_b 1) = S(x) :_b 0$
[b6]	$x + 0 = x$
[b7]	$0 + x = x$
[b8]	$x + 1 = S(x)$
[b9]	$1 + x = S(x)$
$[\text{b10}.i.j]_{i,j=0}^{1}$	$(x :_b i) + (y :_b j) = ((x + y) :_b i) + j$
[b11]	$x \cdot 0 = 0$
[b12]	$x \cdot 1 = x$
$[\text{b13}.i]_{i=0}^{1}$	$x \cdot (y :_b i) = ((x \cdot y) :_b 0) + (x \cdot i)$
$[\text{b14}.i]_{i=1}^{8}$	$i' = S(i)$
$[\text{b15}.i]_{i=0}^{9}$	$x :_d i = (x \cdot S(9)) + i$

Table 6: $\mathbb{N}_{bud}$, natural numbers in binary view

2.2 Binary view

In Table 6 a DDRS for a binary view of natural numbers is displayed, which employs the successor function as an auxiliary function. Leading zeros except for the zero itself are removed by $[\text{b1}.i]_{i=0}^{1}$, and successor terms are rewritten by [b2] – [b5]. This DDRS contains fifteen (parametric) rules (that is, 18 rules for the specification of addition and multiplication, and 18 that serve the rewriting from decimal notation to binary notation via successor terms[1]). In the binary view natural numbers are identified with normal forms in binary notation. The specification has a canonical term algebra $\mathbb{N}_{bud}$ which is isomorphic to the canonical term algebra $\mathbb{N}_{ubd}$ of the specification in Table 1.

In Table 7 minus and predecessor are introduced and the transition from a signature for natural numbers to a signature for integers is made; the rules in this table extend those of Table 6 and define the canonical term algebra $\mathbb{Z}_{bud}$ that is isomorphic to the canonical term algebra $\mathbb{Z}_{ubd}$ of the specification in Table 2. The DDRS thus defined contains 33 (parametric) rules (that is, 36+24 rules in total). We attempt to provide some intuition for equations [b26] and [b27]:

$$(-x) :_b i$$

should be equal to $(-x :_b 0) + i$, so $(-x) :_b 0 = -(x :_b 0)$, and $(-x) :_b 1$ is determined by

$$-(P(x :_b 0)) \overset{[\text{b20}]}{=} -(P(x) :_b 1).$$

[1] Note that there is no rule [b14.0] that is, $1 = S(0)$, because 1 is a normal form in binary view.

Equations [b24] and [b25] can be explained in a similar way:

$$\begin{array}{lll} S(-(x:_b 0)) & \text{should be equal to} & -(P(x:_b 0)) = -(P(x):_b 1), \\ S(-(x:_b 1)) & \text{should be equal to} & -(P(x:_b 1)) = -(x:_b 0). \end{array}$$

The rewrite rules of the DDRS specified by Tables 6 and 7 (viewed as equations) are semantic consequences of the equations for commutative rings (equations (1) – (8) in Table 3).

[b16]	$-0 = 0$
[b17]	$-(-x) = x$
[b18]	$P(0) = -1$
[b19]	$P(1) = 0$
[b20]	$P(x:_b 0) = P(x):_b 1$
[b21]	$P(x:_b 1) = x:_b 0$
[b22]	$P(-x) = -S(x)$
[b23]	$S(-1) = 0$
[b24]	$S(-(x:_b 0)) = -(P(x):_b 1)$
[b25]	$S(-(x:_b 1)) = -(x:_b 0)$
[b26]	$(-x):_b 0 = -(x:_b 0)$
[b27]	$(-x):_b 1 = -(P(x):_b 1)$
[b28]	$x + (-1) = P(x)$
[b29]	$(-1) + x = P(x)$
[b30.$i.j]_{i,j=0}^{1}$	$(x:_b i) + (-(y:_b j)) = ((x + (-y)):_b i) + (-j)$
[b31.$i.j]_{i,j=0}^{1}$	$(-(y:_b j)) + (x:_b i) = ((x + (-y)):_b i) + (-j)$
[b32]	$(-x) + (-y) = -(x + y)$
[b33]	$x \cdot (-y) = -(x \cdot y)$

Table 7: $\mathbb{Z}_{bud}$, integer numbers in binary view, continuation of Table 6

$[\mathrm{d1}.i]_{i=0}^{9}$	$0 :_d i = i$
$[\mathrm{d2}.i]_{i=0}^{8}$	$S(i) = i'$
[d3]	$S(9) = 1 :_d 0$
$[\mathrm{d4}.i]_{i=0}^{8}$	$S(x :_d i) = x :_d i'$
[d5]	$S(x :_d 9) = S(x) :_d 0$
[d6]	$x + 0 = x$
[d7]	$0 + x = x$
$[\mathrm{d8}.i]_{i=0}^{8}$	$x + i' = S(x) + i$
$[\mathrm{d9}.i]_{i=0}^{8}$	$i' + x = S(x) + i$
$[\mathrm{d10}.i.j]_{i,j=0}^{9}$	$(x :_d i) + (y :_d j) = ((x + y) :_d i) + j$
[d11]	$x \cdot 0 = 0$
$[\mathrm{d12}.i]_{i=0}^{8}$	$x \cdot i' = (x \cdot i) + x$
$[\mathrm{d13}.i]_{i=0}^{9}$	$x \cdot (y :_d i) = ((x \cdot y) :_d 0) + (x \cdot i)$
$[\mathrm{d14}.i]_{i=0}^{1}$	$x :_b i = (x + x) + i$

Table 8: $\mathbb{N}_{dub}$, natural numbers in decimal view

2.3 Decimal view

Table 8 defines a DDRS for a decimal view of natural numbers, consisting of fourteen (parametric) rules (172 rules in total). This DDRS defines the datatype $\mathbb{N}_{dub}$ that is isomorphic to the canonical term algebra $\mathbb{N}_{ubd}$ of the specification in Table 1. Leading zeros except for the zero itselfare removed by $[\mathrm{d1}.i]_{i=0}^{9}$, and successor terms are rewritten by $[\mathrm{d2}.i]_{i=0}^{8}$ – [d5]. Rewriting unary notation is part of this DDRS, and the last rule scheme $[\mathrm{d14}.i]_{i=0}^{1}$ serves the rewriting from binary notation.

Before we extend this DDRS to the integers, we define a variant of successor notation for digits that we call "10 minus" subtraction for decimal digits, and we write

$$i^\star$$

for the "10 minus" decimal digit of i. In Figure 2 we display all identities for $i^\star$, and we shall use these in order to cope with terms of the form $(-x) :_d i$ for $i = 1, ..., 9$.

$1^\star \equiv 9$	$4^\star \equiv 6$	$7^\star \equiv 3$
$2^\star \equiv 8$	$5^\star \equiv 5$	$8^\star \equiv 2$
$3^\star \equiv 7$	$6^\star \equiv 4$	$9^\star \equiv 1$

Figure 2: "10 minus" subtraction for decimal digits

[d15]	$-0 = 0$
[d16]	$-(-x) = x$
[d17]	$P(0) = -1$
[d18.i]$_{i=0}^{8}$	$P(i') = i$
[d19]	$P(x :_d 0) = P(x) :_d 9$
[d20.i]$_{i=0}^{8}$	$P(x :_d i') = x :_d i$
[d21]	$P(-x) = -S(x)$
[d22.i]$_{i=0}^{8}$	$S(-i') = -i$
[d23]	$S(-(x :_d 0)) = -(P(x) :_d 9)$
[d24.i]$_{i=0}^{8}$	$S(-(x :_d i')) = -(x :_d i)$
[d25]	$(-x) :_d 0 = -(x :_d 0)$
[d26.i]$_{i=0}^{9}$	$(-x) :_d i = -(P(x) :_d i^{\star})$
[d27.i]$_{i=0}^{8}$	$x + (-i') = P(x) + (-i)$
[d28.i]$_{i=0}^{8}$	$(-i') + x = P(x) + (-i)$
[d29.$i.j$]$_{i,j=0}^{9}$	$(x :_d i) + (-(y :_d j)) = ((x + (-y)) :_d i) + (-j)$
[d30.$i.j$]$_{i,j=0}^{9}$	$(-(y :_d j)) + (x :_d i) = ((x + (-y)) :_d i) + (-j)$
[d31]	$(-x) + (-y) = -(x + y)$
[d32]	$x \cdot (-y) = -(x \cdot y)$

Table 9: $\mathbb{Z}_{dub}$, integers in decimal view, continuation of Table 8 (and using $i^{\star}$ from Figure 2)

In Table 9 minus and predecessor are added and the transition to a signature for integers is made; the rules in this table extend those of Table 8. The DDRS thus defined is named $\mathbb{Z}_{dub}$ and is isomorphic to the canonical term algebra $\mathbb{Z}_{ubd}$ of the specification in Table 2; it contains 32 (parametric) rules (that is, 172 + 273 rules in total).

The (twenty one) equations captured by [d23] – [d26.i]$_{i=0}^{9}$ can be explained in a similar fashion as was done in the previous section for [b24] – [b27]: for example,

$$(-5) :_d 3$$

should be equal to $-(5 :_d 0) + 3 = -(4 :_d 7)$, and this follows immediately from the appropriate equation in [d26.i]$_{i=0}^{9}$.

The rewrite rules of the DDRS specified by Tables 8 and 9 (viewed as equations) are semantic consequences of the equations for commutative rings (equations (1) – (8) in Table 3).

3 Alternative DDRS's for integers with digit tree constructors

Having defined DDRS's that employ (postfix) digit append functions in Section 2, we now consider the more general *digit tree constructor* functions. For the binary view, this approach is followed by Bouma and Walters in [7]; for a view based on any radix, this approach is further continued in Walters [13] and Walters and Zantema [14], where the constructor is called *juxtaposition* because it goes with the absence of a function symbol in order to be close to ordinary decimal and binary notation.

We extend the signature $\Sigma_{\mathbb{Z}}$ defined in Section 1.2 with the following three functions (infix):

$$\hat{u}\,,\ \hat{b}\,,\ \hat{d}\ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z},$$

called "unary digit tree constructor function", "binary digit tree constructor function", and "decimal digit tree constructor function", and to be used for unary, binary notation and decimal notation, respectively. The latter two constructors serve to represent logarithmic notation and satisfy the semantic equations $[\![x \,\hat{b}\, y]\!] = 2 \cdot [\![x]\!] + [\![y]\!]$ and $[\![x \,\hat{d}\, y]\!] = 10 \cdot [\![x]\!] + [\![y]\!]$.

For integer numbers in decimal view or binary view, normal forms are the relevant digits, all applications of the respective constructor with left argument a non-zero normal form and right argument a digit, and all minus instances $-t$ for each such non-zero normal form t, these satisfy $[\![-(t)]\!] = -([\![t]\!])$. E.g.,

$$(9 \,\hat{d}\, 7) \,\hat{d}\, 5 \quad \text{and} \quad ((1 \,\hat{b}\, 0) \,\hat{b}\, 0) \,\hat{b}\, 1$$

represent the decimal number 975, and the binary number 1001, respectively, and $-(((1 \,\hat{b}\, 0) \,\hat{b}\, 0) \,\hat{b}\, 1)$ is the normal form that represents the additional inverse of the latter. A minor complication with decimal and binary digit tree constructors is that we now have to consider rewritings such as

$$2 \,\hat{d}\, (1 \,\hat{d}\, 5) = (2 + 1) \,\hat{d}\, 5 = 3 \,\hat{d}\, 5 \quad (= 35),$$

which perhaps are somewhat non-intuitive. For integers in unary view, thus with unary digit tree constructor, this complication is absent (see Section 3.1).

We keep the presentation of the resulting DDRS's (those defining the binary and decimal view are based on [13, 14]) minimal in the sense that rules for conversion from the one view to the other are left out. Of course, it is easy to define such rules. Rewrite rules for conversion to and from the datatypes defined in Section 2 are also omitted, although such rules are also not difficult to define.

3.1 Unary view with digit tree constructor

For naturals in this particular unary view, normal forms are 0 and expressions $t \,\hat{u}\, 0$ with t a normal form (thus, with association of $\hat{u}$ to the left). Of course, the phenomenon of "removing leading zeros" does not exist in this particular unary view (as in the datatype $\mathbb{N}_{u'}$ defined in Table 4). The resulting datatype $\mathbb{N}_{ut}$ is defined in Table 10.

In the unary view, $\hat{u}$ is an associative operator, as is clear from rule [ut1] (in contrast to digit tree constructors for the binary and decimal case). Moreover, the commutative variants $t \,\hat{u}\, r$ and $r \,\hat{u}\, t$ rewrite to the same normal form. The latter property also follows from the following semantics for closed terms:

$$\begin{aligned}
[\![0]\!] &= 0, \\
[\![x \,\hat{u}\, y]\!] &= [\![x]\!] + [\![y]\!] + 1, \\
[\![x + y]\!] &= [\![x]\!] + [\![y]\!], \\
[\![x \cdot y]\!] &= [\![x]\!] \cdot [\![y]\!].
\end{aligned}$$

[ut1]	$x \mathbin{\hat{u}} (y \mathbin{\hat{u}} z) = (x \mathbin{\hat{u}} y) \mathbin{\hat{u}} z$
[ut2]	$x + 0 = x$
[ut3]	$x + (y \mathbin{\hat{u}} 0) = (x + y) \mathbin{\hat{u}} 0$
[ut4]	$x \cdot 0 = 0$
[ut5]	$x \cdot (y \mathbin{\hat{u}} 0) = (x \cdot y) + x$

Table 10: $\mathbb{N}_{ut}$, natural numbers in unary view with unary digit tree constructor

Observe that

$$x + (y \mathbin{\hat{u}} z) = (x + y) \mathbin{\hat{u}} z \quad \text{and} \quad x \cdot (y \mathbin{\hat{u}} z) = (x \cdot (y + z)) + x$$

are valid equations in $\mathbb{N}_{ut}$.

The extension to integer numbers can be done in a similar fashion as in the previous section, thus obtaining normal forms of the form $-(t)$ with t a non-zero normal form in $\mathbb{N}_{ut}$. However, also terms of the form $x \mathbin{\hat{u}} (-y)$ and variations thereof have to be considered. We define this extension in Table 11 below and call the resulting datatype $\mathbb{Z}_{ut}$.

Adding the interpretation rule $[\![-x]\!] = -[\![x]\!]$ and exploiting the commutativity of $\mathbin{\hat{u}}$ in $[\![x \mathbin{\hat{u}} y]\!]$, it can be easily checked that [ut6] – [ut18] (as equations) are sound. Moreover, the equation

$$(-x) \mathbin{\hat{u}} y = y \mathbin{\hat{u}} (-x)$$

also holds in $\mathbb{Z}_{ut}$.

[ut6]	$-0 = 0$	[ut17]	$x \cdot (-y) = -(x \cdot y)$
[ut7]	$-(-x) = x$	[ut18]	$(-x) \cdot y = -(x \cdot y)$
[ut8]	$0 \mathbin{\hat{u}} (-(x \mathbin{\hat{u}} 0)) = -x$		
[ut9]	$(x \mathbin{\hat{u}} 0) \mathbin{\hat{u}} (-(y \mathbin{\hat{u}} 0)) = x \mathbin{\hat{u}} (-y)$		
[ut10]	$(-(x \mathbin{\hat{u}} 0)) \mathbin{\hat{u}} 0 = -x$		
[ut11]	$(-(y \mathbin{\hat{u}} 0)) \mathbin{\hat{u}} (x \mathbin{\hat{u}} 0) = x \mathbin{\hat{u}} (-y)$		
[ut12]	$(-(x \mathbin{\hat{u}} 0)) \mathbin{\hat{u}} (-(y \mathbin{\hat{u}} 0)) = -((x + y) \mathbin{\hat{u}} 0)$		
[ut13]	$0 + x = x$		
[ut14]	$(x \mathbin{\hat{u}} 0) + (-(y \mathbin{\hat{u}} 0)) = x + (-y)$		
[ut15]	$(-(y \mathbin{\hat{u}} 0)) + (x \mathbin{\hat{u}} 0) = x + (-y)$		
[ut16]	$(-x) + (-y) = -(x + y)$		

Table 11: $\mathbb{Z}_{ut}$, integer numbers in unary view with unary digit tree constructor, continuation of Table 10

[bt1]	$0 \mathbin{\hat{b}} x = x$	[bt8]	$x \cdot 0 = 0$
[bt2]	$x \mathbin{\hat{b}} (y \mathbin{\hat{b}} z) = (x + y) \mathbin{\hat{b}} z$	[bt9]	$x \cdot 1 = x$
[bt3]	$0 + x = x$	[bt10]	$x \cdot (y \mathbin{\hat{b}} z) = (x \cdot y) \mathbin{\hat{b}} (x \cdot z)$
[bt4]	$1 + 0 = 1$		
[bt5]	$1 + 1 = 1 \mathbin{\hat{b}} 0$		
[bt6]	$1 + (x \mathbin{\hat{b}} y) = x \mathbin{\hat{b}} (1 + y)$		
[bt7]	$(x \mathbin{\hat{b}} y) + z = x \mathbin{\hat{b}} (y + z)$		

Table 12: $\mathbb{N}_{bt}$, natural numbers in binary view with binary digit tree constructor

3.2 Binary view with digit tree constructor

For naturals in binary view with the binary digit tree constructor, the associated datatype $\mathbb{N}_{bt}$ is defined in Table 12. The rewrite system defined by [bt1] – [bt7] (thus excluding multiplication) is taken from [7], in which it was proven confluent and terminating.

In [14] a rewrite system for integer arithmetic is provided with next to juxtaposition and (unary) minus also addition and subtraction and multiplication, and proven ground confluent and terminating with respect to any radix using rule schemata (that is, parametric rules). In Table 13 we present a variant of this rewrite system without subtraction for the binary digit tree constructor, and define the datatype $\mathbb{Z}_{bi}$. Because binary view requires so few digits and because we will follow another approach for the decimal view, we have no use for rule schemata and just define all relevant instances ([bi5], [bi10], [bi15] – [bi17], and [bi20] – [bi21]).

[bi1]	$0 \mathbin{\hat{b}} x = x$	[bi13]	$-0 = 0$
[bi2]	$x \mathbin{\hat{b}} (y \mathbin{\hat{b}} z) = (x + y) \mathbin{\hat{b}} z$	[bi14]	$-(-x) = x$
[bi3]	$0 + x = x$	[bi15]	$1 \mathbin{\hat{b}} (-1) = 1$
[bi4]	$x + 0 = x$	[bi16]	$(x \mathbin{\hat{b}} 0) \mathbin{\hat{b}} (-1) = (x \mathbin{\hat{b}} (-1)) \mathbin{\hat{b}} 1$
[bi5]	$1 + 1 = 1 \mathbin{\hat{b}} 0$	[bi17]	$(x \mathbin{\hat{b}} 1) \mathbin{\hat{b}} (-1) = (x \mathbin{\hat{b}} 0) \mathbin{\hat{b}} 1$
[bi6]	$x + (y \mathbin{\hat{b}} z) = y \mathbin{\hat{b}} (x + z)$	[bi18]	$x \mathbin{\hat{b}} (-(y \mathbin{\hat{b}} z)) = -((y + (-x)) \mathbin{\hat{b}} z)$
[bi7]	$(x \mathbin{\hat{b}} y) + z = x \mathbin{\hat{b}} (y + z)$	[bi19]	$(-x) \mathbin{\hat{b}} y = -(x \mathbin{\hat{b}} (-y))$
[bi8]	$x \cdot 0 = 0$	[bi20]	$1 + (-1) = 0$
[bi9]	$0 \cdot x = 0$	[bi21]	$(-1) + 1 = 0$
[bi10]	$1 \cdot 1 = 1$	[bi22]	$x + (-(y \mathbin{\hat{b}} z)) = -(y \mathbin{\hat{b}} (z + (-x)))$
[bi11]	$x \cdot (y \mathbin{\hat{b}} z) = (x \cdot y) \mathbin{\hat{b}} (x \cdot z)$	[bi23]	$(-(x \mathbin{\hat{b}} y)) + z = -(x \mathbin{\hat{b}} (y + (-z)))$
[bi12]	$(x \mathbin{\hat{b}} y) \cdot z = (x \cdot z) \mathbin{\hat{b}} (y \cdot z)$	[bi24]	$x \cdot (-y) = -(x \cdot y)$
		[bi25]	$(-x) \cdot y = -(x \cdot y)$

Table 13: $\mathbb{Z}_{bi}$, integer numbers in binary view with binary digit tree constructor

[dt1]	$0 \mathbin{\hat{d}} x = x$
[dt2]	$x \mathbin{\hat{d}} (y \mathbin{\hat{d}} z) = (x + y) \mathbin{\hat{d}} z$
$[\text{dt3}.i]_{i=0}^{8}$	$S(i) = i'$
[dt4]	$S(9) = 1 \mathbin{\hat{d}} 0$
$[\text{dt5}.i]_{i=0}^{8}$	$S(x \mathbin{\hat{d}} i) = x \mathbin{\hat{d}} i'$
[dt6]	$S(x \mathbin{\hat{d}} 9) = S(x) \mathbin{\hat{d}} 0$
[dt7]	$x + 0 = x$
$[\text{dt8}.i]_{i=0}^{8}$	$x + i' = S(x) + i$
$[\text{dt9}.i]_{i=0}^{9}$	$x + (y \mathbin{\hat{d}} i) = (y \mathbin{\hat{d}} x) + i$
[dt10]	$x \cdot 0 = 0$
$[\text{dt11}.i]_{i=0}^{8}$	$x \cdot i' = x + (x \cdot i)$
$[\text{dt12}.i]_{i=0}^{9}$	$x \cdot (y \mathbin{\hat{d}} i) = ((x \cdot y) \mathbin{\hat{d}} 0) + (x \cdot i)$

Table 14: $\mathbb{N}_{dt}$, natural numbers with decimal digit tree constructor in decimal view (using i' from Figure 1)

3.3 Decimal view with digit tree constructor

For naturals in decimal view with the decimal digit tree constructor, we make use of successor terms, in order to avoid (non-parametric) rules such as

$$1 + 1 = 2, ..., \quad 9 + 8 = 1 \mathbin{\hat{d}} 7, \quad 9 + 9 = 1 \mathbin{\hat{d}} 8,$$
$$1 \cdot 1 = 1, ..., \quad 8 \cdot 9 = 7 \mathbin{\hat{d}} 2, \quad 9 \cdot 9 = 8 \mathbin{\hat{d}} 1.$$

The associated datatype $\mathbb{N}_{dt}$ is defined in Table 14. Note that rules of the form

$$i' + x = i + S(x)$$

instead of (or next to) $[\text{dt8}.i]_{i=0}^{8}$ would destroy termination: $2 + 1 \mapsto 1 + S(1) \mapsto S(1) + 1 \mapsto 2 + 1$. Moreover, the interplay between digit tree constructor, successor and normal form notation makes it by rules $[\text{dt9}.i]_{i=0}^{9}$ possible not to incorporate the rewrite rule $0 + x = x$ in this particular, relatively simple rewrite system.

The extension to integers is given by the rules in Table 15, which define the datatype $\mathbb{Z}_{dt}$. In contrast to the approaches in [13, 14] with juxtaposition, we now make use of both successor terms and predecessor terms, and the rewrite system presented here is composed from rewrite rules for successor and predecessor, rewrite rules defined in [13, 14], and combinations thereof. Note that we can still do without the rewrite rule $0 + x = x$:

$0 + (-0) = 0$	by [dt13] and [dt7],
$0 + (-i') = P(0) + (-i) = ... = -i'$	by $[\text{dt27}.i]_{i=0}^{8}$ (and some more rules),
$0 + (-(t_1 \mathbin{\hat{d}} t_2)) = -(t_1 \mathbin{\hat{d}} t_2)$	by rules [dt28], [dt13], and [dt7].

[dt1]	$0 \mathbin{\hat{d}} x = x$	[dt13]	$-0 = 0$
[dt2]	$x \mathbin{\hat{d}} (y \mathbin{\hat{d}} z) = (x + y) \mathbin{\hat{d}} z$	[dt14]	$-(-x) = x$
$[\text{dt3}.i]_{i=0}^{8}$	$S(i) = i'$	[dt15]	$P(0) = -1$
[dt4]	$S(9) = 1 \mathbin{\hat{d}} 0$	$[\text{dt16}.i]_{i=0}^{8}$	$P(i') = i$
$[\text{dt5}.i]_{i=0}^{8}$	$S(x \mathbin{\hat{d}} i) = x \mathbin{\hat{d}} i'$	[dt17]	$P(x \mathbin{\hat{d}} 0) = P(x) \mathbin{\hat{d}} 9$
[dt6]	$S(x \mathbin{\hat{d}} 9) = S(x) \mathbin{\hat{d}} 0$	$[\text{dt18}.i]_{i=0}^{8}$	$P(x \mathbin{\hat{d}} i') = x \mathbin{\hat{d}} i$
[dt7]	$x + 0 = x$	[dt19]	$P(-x) = -S(x)$
$[\text{dt8}.i]_{i=0}^{8}$	$x + i' = S(x) + i$	$[\text{dt20}.i]_{i=0}^{8}$	$S(-i') = -i$
$[\text{dt9}.i]_{i=0}^{9}$	$x + (y \mathbin{\hat{d}} i) = (y \mathbin{\hat{d}} x) + i$	[dt21]	$S(-(x \mathbin{\hat{d}} 0)) = -(P(x) \mathbin{\hat{d}} 9)$
[dt10]	$x \cdot 0 = 0$	$[\text{dt22}.i]_{i=0}^{8}$	$S(-(x \mathbin{\hat{d}} i')) = -(x \mathbin{\hat{d}} i)$
$[\text{dt11}.i]_{i=0}^{8}$	$x \cdot i' = x + (x \cdot i)$	[dt23]	$(-x) \mathbin{\hat{d}} y = -(x \mathbin{\hat{d}} (-y))$
$[\text{dt12}.i]_{i=0}^{9}$	$x \cdot (y \mathbin{\hat{d}} i) = ((x \cdot y) \mathbin{\hat{d}} 0) + (x \cdot i)$	$[\text{dt24}i.j]_{i,j=1}^{9}$	$i \mathbin{\hat{d}} (-j) = P(i) \mathbin{\hat{d}} j^{\star}$
		$[\text{dt25}.i]_{i=1}^{9}$	$(x \mathbin{\hat{d}} y) \mathbin{\hat{d}} (-i) = P(x \mathbin{\hat{d}} y) \mathbin{\hat{d}} i^{\star}$
		[dt26]	$x \mathbin{\hat{d}} (-(y \mathbin{\hat{d}} z)) = -((y + (-x)) \mathbin{\hat{d}} z)$
		$[\text{dt27}.i]_{i=0}^{8}$	$x + (-i') = P(x) + (-i)$
		[dt28]	$x + (-(y \mathbin{\hat{d}} z)) = -(y \mathbin{\hat{d}} (z + (-x)))$
		$[\text{dt29}.i]_{i=0}^{8}$	$(-i') + x = P(x) + (-i)$
		[dt30]	$(-(x \mathbin{\hat{d}} y)) + z = -(x \mathbin{\hat{d}} (y + (-z)))$
		[dt31]	$x \cdot (-y) = -(x \cdot y)$
		[dt32]	$(-x) \cdot y = -(x \cdot y)$

Table 15: $\mathbb{Z}_{dt}$, integer numbers with decimal digit tree constructor in decimal view (using i' from Figure 1 and $i^{\star}$ from Figure 2)

4 Concluding remarks

This paper is about the design (by means of trial and error) of datatype defining rewrite systems rather than about the precise analysis of the various rewrite systems per se. Termination proofs, Knuth-Bendix completion, full confluence proofs, or merely ground confluence proofs, are postponed until a stable design direction has been developed for a sequence of datatypes. What matters in addition to readability and conciseness of each DDRS is at this stage a reasonable confidence that each of these rewrite systems is strongly terminating and ground confluent and that the intended normal forms are reached by means of rewriting.

When specifying a datatype of integers as an extension of the naturals, the unary view leads to satisfactory results, but with high inefficiency. For the binary view and the decimal view corresponding extensions are possible (and provided), but the resulting rewrite systems are at first sight significantly less concise and comprehensible. Some further remarks:

1. The three DDRS's (datatype defining rewrite systems) for integers given in Section 2 each produce an extension datatype for a datatype for the natural numbers. An initial algebra specification of the datatype of integers is obtained from any of the DDRS's given in [1] by

   - taking the reduct to the signature involving unary, binary, and decimal notation only,
   - removing rewrite rules involving operators for hexadecimal notation,
   - expanding the signature with a unary additive inverse and a unary predecessor function,
   - adding rewrite rules (in equational form) that allow for the unique normalization of closed terms involving the minus sign,

   while making sure that these rewrite rules (viewed as equations) are semantic consequences of the equations for commutative rings.

2. Syntax for hexadecimal notation has been omitted because that usually plays no role when dealing with integers. It is an elementary exercise to incorporate hexadecimal notation.

3. The DDRS's for the binary view and the decimal view are hardly intelligible unless one knows that the objective is to construct a commutative ring. A decimal normal form is defined as either a digit, or an application of a decimal append function $_ :_d i$ to a non-zero normal form (for all digits i). This implies the absence of (superfluous) leading zeros, and the (ground) normal forms thus obtained correspond bijectively to the non-negative integers (that is, $\mathbb{N}$). Incorporating all minus instances $-(t)$ for each non-zero normal form t yields the class of normal forms. The "semantics" of these normal forms in the language of commutative rings is very simple:

$$\begin{aligned}
[\![0]\!] &= 0, \\
[\![i']\!] &= [\![i]\!] + 1 \quad \text{for all digits } i \text{ (and } i' \text{ defined as in Fig. 1)}, \\
[\![x :_d i]\!] &= (\underline{10} \cdot [\![x]\!]) + [\![i]\!] \quad \text{for all digits } i \text{ and } \underline{10} = [\![9]\!] + 1, \\
[\![-(x)]\!] &= -([\![x]\!]).
\end{aligned}$$

   A binary normal form has similar semantics: $[\![x :_b i]\!] = (\underline{2} \cdot [\![x]\!]) + [\![i]\!]$ for digits 0, 1, and $\underline{2} = 1 + 1$.

4. Understanding the concept of a commutative ring can be expected only from a person who has already acquired an understanding of the structure of integers and who accepts the concept of generalization of a structure to a class of structures sharing some but not all of its properties.

   In other words, the understanding that a DDRS for the integers is provided in the binary view and in the decimal view can only be communicated to an audience under the assumption that a reliable mental picture of the integers already exists in the minds of members of the audience. This mental picture, however, can in principle be communicated by taking notice of the DDRS for the unary view first.

   This conceptual (near) circularity may be nevertheless be considered a significant weakness of the approach of defining (and even introducing) the integers as an extension of naturals by means of rewriting.

In Section 3 we discussed some alternatives for these DDRS's based on papers of Bouma and Walters [7], Walters [13], and Walters and Zantema [14] in which digit tree constructors are employed. In this case, a digit is a normal form, and so is an application of the digit tree constructor that adheres to association to the left and with the removal of (superfluous) leading zeros. Thus, $n \mathbin{\hat{d}} i$ is a normal form if n is a non-zero normal form and i a digit. Incorporating all minus instances $-(t)$ for each non-zero normal form t yields the class of normal forms for integers.

Of course, a decimal notation as 689 is so common that one usually does not question whether it represents $(6 :_d 8) :_d 9$ or $(6 \hat{}_d 8) \hat{}_d 9$ or some other formally defined notation. Nevertheless, as we have seen, different algorithmic approaches to for example addition may apply, although one would preferably not hamper an (initial) arithmetical method with notation such as $x \hat{}_d (y \hat{}_d z)$ and rewrite rules such as $x \hat{}_d (y \hat{}_d z) = (x + y) \hat{}_d z$ and those for +, and for this reason we have a preference for the DDRS's defined in Section 2 (and in the first version of this report).

In [13] a ground-complete term rewriting system (TRS) based on juxtaposition for integer arithmetic with addition and subtraction is presented; this system is proven ground confluent and terminating with respect to any radix. In [14], the authors extend this system with multiplication and prove ground-completeness, using *semantic labelling* for their termination proof, and judge this rewrite system to have good efficiency and readability (in comparison with some alternatives discussed in that paper). Furthermore, the authors also discuss a TRS that is based on successor and predecessor notation, and in which minus is not used: negative numbers are represented by normal forms $P(0)$, $P(P(0))$, and so on. This rewrite system is comparable to the DDRS in Table 2 that defines $\mathbb{Z}_{ubd}$ and is proven confluent and terminating, and judged to have poor readability and (too) high complexity. Finally, in [14], the authors also provide a complete TRS for natural numbers with addition and multiplication that is based on digit append.

We briefly mention two other, comparable approaches to integer arithmetic that are also based on some form of digit append constructors for representing integer numbers. In [11], Kennaway defines integer arithmetic for any base $b > 1$ (integers are represented as lists with appends to the left), using extra digits b to $2(b - 1)$ and also some auxiliary functions. This results in a complete TRS, of which a proper subclass of the normal forms represents the integer numbers; moreover, this TRS allows for any total recursive function an immediate extension in which also that function is represented while completeness is preserved. Secondly, in [9], Contejean, Marché and Rabehasaina introduce integer arithmetic based on *balanced ternary numbers*, that is, numbers that can be represented by a digit append function $:_t$ with digits -1,0,1 and semantics $[\![i]\!] = i$ and $[\![x :_t i]\!] = \underline{3} \cdot [\![x]\!] + i$ (see, e.g., Knuth [12]) and provide a TRS that is confluent and terminating modulo associativity and commutativity of addition and multiplication.

Based on either a DDRS for the natural numbers or a DDRS for the integers one may develop a DDRS for rational numbers in various ways. It is plausible to consider the meadow of rational numbers of [6] or the non-involutive meadow of rational numbers (see [2]) or the common meadow of rational numbers (see [3]) as abstract algebraic structures for rationals in which unary, binary, and decimal notation are to be incorporated in ways possibly based on the specifications presented above. Furthermore, one does well to consider the work discussed in [9] on a term rewriting system for rational numbers, in which arithmetic for rational numbers is specified (this is the main result in [9], for which the above-mentioned work on integer arithmetic is a preliminary): the authors specify rational numbers by means of a TRS that is complete modulo associativity and commutativity of addition and multiplication, taking advantage of Stein's algorithm for computing gcd's of non-negative integers without any division[2] (see, e.g., [12]).

A survey of equational algebraic specifications for abstract datatypes is provided in [15]. In [5] one finds the general result that computable abstract datatypes can be specified by means of specifications which are confluent and strongly terminating term rewriting systems. Some general results on algebraic specifications can be found in [8, 4, 10]. More recent applications of equational specifications can be found in [6].

[2]Apart from halving even numbers, which is easy in binary notation, but can otherwise be specified with a shift operation.

References

[1] Bergstra, J.A. (2014). Four datatype defining rewrite systems for an abstract datatype of natural numbers. *University of Amsterdam, Informatics Institute, Section Theory of Computer Science, Research Report TCS1407v2*, `http://www.science.uva.nl/pub/programming-research/tcsreports/TCS1407v2.pdf`.

[2] Bergstra, J.A. and Middelburg, C.A. (2014, June 9). Division by zero in non-involutive meadows. Available at `http://arxiv.org/abs/1406.2092v1` [math.RA].

[3] Bergstra, J.A. and Ponse, A. (2014, June 26). Division by zero in common meadows. Available at `http://arxiv.org/abs/1406.6878v1` [math.RA].

[4] Bergstra, J.A. and Tucker, J.V. (1987). Algebraic specifications of computable and semicomputable data types. *Theoretical Computer Science*, 50(2):137–181.

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

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

[7] Bouma, L.G. and Walters, H.R. (1989). Implementing algebraic specifications. In J.A. Bergstra, J. Heering, and P. Klint (Eds.), *Algebraic Specification* (Chapter 5), Addison-Wesley, pp. 199–282.

[8] Broy, M., Wirsing, M., and Pair, C. (1984). A systematic study of models of abstract data types. *Theoretical Computer Science,* 33(2):139–174.

[9] Contejean, E., Marché, C., and Rabehasaina, L. (1997). Rewrite systems for natural, integral, and rational arithmetic. In H. Comon (Ed.), *Rewriting Techniques and Applications (Proceedings 8th International Conference, RTA'97)*, Lecture Notes in Computer Science, Vol. 1232, Springer, pp. 98–112.

[10] Gaudel, M.-C. and James, P.R. (1998). Testing algebraic data types and processes: a unifying theory. *Formal Aspects of Computing,* 10(5-6):436–451.

[11] Kennaway, J.R. (1995). Complete term rewrite systems for decimal arithmetic and other total recursive functions. 2nd International Workshop on Termination, La Bresse, 29-31 May 1995 (available at `http://citeseerx.ist.psu.edu`).

[12] Knuth, D.E. (1997). The Art of Computer Programming, Volume 2 (3rd edition.): Seminumerical Algorithms. Addison-Wesley.

[13] Walters, H.R. (1994). A complete term rewriting system for decimal integer arithmetic. Report CS-R9435, CWI, Amsterdam. `http://oai.cwi.nl/oai/asset/5140/5140D.pdf`

[14] Walters, H.R. and Zantema, H. (1995). Rewrite systems for integer arithmetic. In J. Hsiang (Ed.), *Rewriting Techniques and Applications (Proceedings 6th International Conference, RTA'95)*, Lecture Notes in Computer Science, Vol. 914, Springer, pp. 324–338.

[15] Wirsing, M. (1991). Algebraic Specification. In: *Handbook of Theoretical Computer Science*, (vol. B), MIT Press, pp. 675–788.

Electronic Reports Series of section Theory of Computer Science

Within this series the following reports appeared.

[TCS1410] J.A. Bergstra and A. Ponse, *Division by Zero in Common Meadows,* section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1407v3] J.A. Bergstra, *Four Complete Datatype Defining Rewrite Systems for an Abstract Datatype of Natural Numbers (version 3),* section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1409] J.A. Bergstra and A. Ponse, *Three Datatype Defining Rewrite Systems for Datatypes of Integers each extending a Datatype of Naturals,* section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1406v3] J.A. Bergstra, *Bitcoin and Islamic Finance (version 3),* section Theory of Computer Science - University of Amsterdam, 2014.

[TCS1407v2] J.A. Bergstra, *Four Complete Datatype Defining Rewrite Systems for an Abstract Datatype of Natural Numbers (version 2),* section Theory of Computer Science - University of Amsterdam, 2014.

[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/