University of Amsterdam
Theory of Computer Science

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

J.A. Bergstra
A. Ponse

Report TCS1409 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

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

June 12, 2014

Abstract

Integer arithmetic is specified according to three views: unary, binary, and decimal notation. In each case we find 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.

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

Contents

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] and it constitutes a further stage in the development of a family of arithmetical data types with corresponding specifications. The resulting specifications (DDRSs) 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 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 3) is entirely adequate, whereas the subsequent specifications for the *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 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 data type 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 Table 1 the successor notation on digits is specified as a transformation of syntax, and we adopt this notation throughout the paper.

Furthermore, for $n, m \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $n < m$, the notation

$$\bigwedge_{i=n}^{m} [t = r]$$

represents the set of rewrite rules $t = r$ with i instantiated from n to m.

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

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

1.2 A signature for integers

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

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

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

   ("decimal append zero", ...,"decimal append nine"); these functions will be used for decimal notation.

The "binary append" and "decimal append" functions can be viewed as instantiations of more general two-place functions "binary digit append" and "decimal digit append", but that would require the introduction of sorts for bits (binary digits) and for digits (decimal digits). This is why we instantiate such "digit append" functions per digit to unary functions.

We will use *postfix notation* for applications of the "digit append" functions, e.g.

$$(9 \oslash 7) \oslash 5 \quad \text{and} \quad ((1 \ominus 0) \ominus 0) \ominus 1$$

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

For the unary view normal the normal forms are those of the unary view on naturals, that is

$$0, S(0), S(S(0)), ...,$$

and all minus instances $-t$ for each non-zero normal form t, e.g. $-(S(S(0)))$. Similarly for the binary view and for the decimal view, normal forms are obtained by extending the respective sets of normal forms with $-t$ for each such normal form t that differs from 0. Thus $-((9 \oslash 7) \oslash 5)$ is an example of a normal form in decimal view, and so is $-((1 \ominus 0) \ominus 0) \ominus 1$ in binary view.

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

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

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 2 provides a DDRS for the natural numbers. Minus and predecessor are absent in this datatype. Successor terms, that is expressions involving zero and successor only, serve as normal forms.

In Table 3 an algebraic specification is provided of the integers numbers with constants zero and one, and with successor, predecessor, addition, and multiplication. This specification extends that of Table 2. 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 (21) is sufficient, and why addition is defined by recursion on both its arguments and also requires (18).

In Table 4 one finds a listing of equations that are true in the datatype $\mathbb{Z}_{ubd}$ that is specified by the DDRS of Table 3. This ensures that these rewrite rules (viewed as equations) are semantic consequences of the equations for commutative rings.

Binary notation and decimal notation are defined by expanding terms into successor terms. This expansion involves a combinatorial explosion in size. That explosion renders the specification in Tables 2 and 3 irrelevant as term rewrite systems from which an efficient implementation can be generated.

In the next section 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 DDRS the quality improves because normal forms are smaller and are reached in fewer rewriting steps.

$$-0 = 0 \tag{8}$$
$$-(-x) = x \tag{9}$$
$$P(0) = -S(0) \tag{10}$$
$$P(S(x)) = x \tag{11}$$
$$P(-x) = -S(x) \tag{12}$$
$$S(-(S(x))) = -x \tag{13}$$
$$x + 0 = x \tag{14}$$
$$0 + x = 0 \tag{15}$$
$$x + S(y) = S(x + y) \tag{16}$$
$$S(x) + y = S(x + y) \tag{17}$$
$$(-x) + (-y) = -(x + y) \tag{18}$$
$$x \cdot 0 = 0 \tag{19}$$
$$x \cdot S(y) = (x \cdot y) + x \tag{20}$$
$$x \cdot (-y) = -(x \cdot y) \tag{21}$$
$$\bigwedge_{i=0}^{8} [\, i' = S(i) \,] \tag{22}$$
$$\bigwedge_{i=0}^{1} [\, x \ominus i = (x \cdot S(1)) + i \,] \tag{23}$$
$$\bigwedge_{i=0}^{9} [\, x \oslash i = (x \cdot S(9)) + i \,] \tag{24}$$

Table 3: Integers in unary view ($\mathbb{Z}_{ubd}$)

$$x + (y + z) = (x + y) + z \tag{25}$$
$$x + y = y + x \tag{26}$$
$$x + 0 = x \tag{27}$$
$$x + (-x) = 0 \tag{28}$$
$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \tag{29}$$
$$x \cdot y = y \cdot x \tag{30}$$
$$1 \cdot x = x \tag{31}$$
$$x \cdot (y + z) = (x \cdot y) + (x \cdot z) \tag{32}$$
$$x \ominus 0 = x + x \tag{33}$$
$$x \ominus 1 = (x + x) + 1 \tag{34}$$

Table 4: Equations valid in $\mathbb{Z}_{ubd}$, where (25) – (32) axiomatize commutative rings

$$\bigwedge_{i=0}^{1}[0 \ominus i = i] \tag{35}$$

$$S(0) = 1 \tag{36}$$
$$S(1) = 1 \ominus 0 \tag{37}$$
$$S(x \ominus 0) = x \ominus 1 \tag{38}$$
$$S(x \ominus 1) = S(x) \ominus 0 \tag{39}$$

$$x + 0 = x \tag{40}$$
$$0 + x = x \tag{41}$$
$$x + 1 = S(x) \tag{42}$$
$$1 + x = S(x) \tag{43}$$
$$(x \ominus 0) + (y \ominus 0) = (x + y) \ominus 0 \tag{44}$$
$$(x \ominus 0) + (y \ominus 1) = (x + y) \ominus 1 \tag{45}$$
$$(x \ominus 1) + (y \ominus 0) = (x + y) \ominus 1 \tag{46}$$
$$(x \ominus 1) + (y \ominus 1) = S(x + y) \ominus 0 \tag{47}$$

$$x \cdot 0 = 0 \tag{48}$$
$$x \cdot 1 = x \tag{49}$$
$$\bigwedge_{i=0}^{1}[x \cdot (y \ominus i) = (x \cdot y) \ominus 0 + i] \tag{50}$$

$$\bigwedge_{i=0}^{8}[i' = S(i)] \tag{51}$$
$$\bigwedge_{i=0}^{9}[x \oslash i = (x \cdot S(9)) + i] \tag{52}$$

Table 5: Naturals in binary view ($\mathbb{N}_{bud}$)

2.2 Binary view

In Table 5 a specification for a binary view of natural numbers is displayed. 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 2.

In Table 6 minus and predecessor are introduced and the transition from a signature for natural numbers to a signature for integers is made. We attempt to provide some intuition for equations (63) and (64):

$$-(x) \ominus i$$

should be equal to $-(x \ominus 0) + i$, so $-(x) \ominus 0 = -(x \ominus 0)$, and $(-x) \ominus 1$ is determined by $-(P(x \ominus 0)) \overset{(57)}{=} -(P(x) \ominus 1)$.

Equations (61) and (62) can be explained in a similar way: $S(-(x \ominus 0))$ should be equal to $-(P(x \ominus 0)) = -(P(x) \ominus 1)$, and $S(-(x \ominus 1))$ should be equal to $-(P(x \ominus 1)) = -(x \ominus 0)$.

$$-0 = 0 \tag{53}$$
$$-(-x) = x \tag{54}$$
$$P(0) = -1 \tag{55}$$
$$P(1) = 0 \tag{56}$$
$$P(x \ominus 0) = P(x) \ominus 1 \tag{57}$$
$$P(x \ominus 1) = x \ominus 0 \tag{58}$$
$$P(-x) = -S(x) \tag{59}$$
$$S(-1) = 0 \tag{60}$$
$$S(-(x \ominus 0)) = -(P(x) \ominus 1) \tag{61}$$
$$S(-(x \ominus 1)) = -(x \ominus 0) \tag{62}$$
$$(-x) \ominus 0 = -(x \ominus 0) \tag{63}$$
$$(-x) \ominus 1 = -(P(x) \ominus 1) \tag{64}$$
$$x + (-1) = P(x) \tag{65}$$
$$(-1) + x = P(x) \tag{66}$$
$$(x \ominus 0) + (-(y \ominus 0)) = (x + (-y)) \ominus 0 \tag{67}$$
$$(x \ominus 0) + (-(y \ominus 1)) = P(x + (-y)) \ominus 1 \tag{68}$$
$$(x \ominus 1) + (-(y \ominus 0)) = (x + (-y)) \ominus 1 \tag{69}$$
$$(x \ominus 1) + (-(y \ominus 1)) = (x + (-y)) \ominus 0 \tag{70}$$
$$(-(y \ominus 0)) + (x \ominus 0) = (x + (-y)) \ominus 0 \tag{71}$$
$$(-(y \ominus 0)) + (x \ominus 1) = (x + (-y)) \ominus 1 \tag{72}$$
$$(-(y \ominus 1)) + (x \ominus 0) = P(x + (-y)) \ominus 1 \tag{73}$$
$$(-(y \ominus 1)) + (x \ominus 1) = (x + (-y)) \ominus 0 \tag{74}$$
$$(-x) + (-y) = -(x + y) \tag{75}$$
$$x \cdot (-y) = -(x \cdot y) \tag{76}$$

Table 6: Additional rules for integers in binary view ($\mathbb{Z}_{bud}$), continuation of Table 5

$$\bigwedge_{i=0}^{9}[0 \oslash i = i] \tag{77}$$

$$\bigwedge_{i=0}^{8}[S(i) = i'] \tag{78}$$

$$S(9) = 1 \oslash 0 \tag{79}$$

$$\bigwedge_{i=0}^{8}[S(x \oslash i) = x \oslash i'] \tag{80}$$

$$S(x \oslash 9) = S(x) \oslash 0 \tag{81}$$

$$x + 0 = x \tag{82}$$

$$0 + x = x \tag{83}$$

$$\bigwedge_{i=0}^{8}[x + i' = S(x) + i] \tag{84}$$

$$\bigwedge_{i=0}^{8}[i' + x = S(x) + i] \tag{85}$$

$$\bigwedge_{i=0}^{9}[\bigwedge_{j=0}^{9}[(x \oslash i) + (y \oslash j) = ((x + y) \oslash i) + j]] \tag{86}$$

$$x \cdot 0 = 0 \tag{87}$$

$$\bigwedge_{i=0}^{8}[x \cdot i' = (x \cdot i) + x] \tag{88}$$

$$\bigwedge_{i=0}^{9}[x \cdot (y \oslash i) = (x \cdot y) \oslash 0 + (x \cdot i)] \tag{89}$$

$$\bigwedge_{i=0}^{1}[x \ominus i = (x + x) + i] \tag{90}$$

Table 7: Natural numbers with addition and multiplication in decimal view ($\mathbb{N}_{dub}$)

2.3 Decimal view

In Table 7 a DDRS for a decimal view of natural numbers is displayed. Of course, successor terms are to be rewritten in this case.

Table 9 contains an extension of the same DDRS to the case of integers. In order to cope with terms of the form $(-x) \oslash i$ with $i = 1, ..., 9$ we define

$$i^{\star}$$

as the "10 minus" digit of i in Table 8.

The (twenty) equations captured by (99) – (102) can be explained in a similar fashion as was done in the previous section for (61) – (64): for example,

$$(-5) \oslash 3$$

should be equal to $-(5 \oslash 0) + 3 = -(4 \oslash 7)$, and this follows immediately from the appropriate equation in (102).

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

Table 8: “10 minus” subtraction for decimal digits

$$-0 = 0 \tag{91}$$
$$-(-x) = x \tag{92}$$
$$P(0) = -1 \tag{93}$$
$$\bigwedge_{i=0}^{8} [\, P(i') = i \,] \tag{94}$$
$$P(x \oslash 0) = P(x) \oslash 9 \tag{95}$$
$$\bigwedge_{i=0}^{8} [\, P(x \oslash i') = x \oslash i \,] \tag{96}$$
$$P(-x) = -S(x) \tag{97}$$
$$\bigwedge_{i=0}^{8} [\, S(-i') = -i \,] \tag{98}$$
$$S(-(x \oslash 0)) = -(P(x) \oslash 9) \tag{99}$$
$$\bigwedge_{i=0}^{8} [\, S(-(x \oslash i')) = -(x \oslash i) \,] \tag{100}$$
$$(-x) \oslash 0 = -(x \oslash 0) \tag{101}$$
$$\bigwedge_{i=1}^{9} [\, (-x) \oslash i = -(P(x) \oslash i^\star) \,] \tag{102}$$
$$\bigwedge_{i=0}^{8} [\, x + (-i') = P(x) + (-i) \,] \tag{103}$$
$$\bigwedge_{i=0}^{8} [\, (-i') + x = P(x) + (-i) \,] \tag{104}$$
$$\bigwedge_{i=0}^{9} [\, \bigwedge_{j=0}^{9} [\, (x \oslash i) + (-(y \oslash j)) = ((x + (-y)) \oslash i) + (-j) \,] \,] \tag{105}$$
$$\bigwedge_{i=0}^{9} [\, \bigwedge_{j=0}^{9} [\, (-(y \oslash j)) + (x \oslash i) = ((x + (-y)) \oslash i) + (-j) \,] \,] \tag{106}$$
$$(-x) + (-y) = -(x + y) \tag{107}$$
$$x \cdot (-y) = -(x \cdot y) \tag{108}$$

Table 9: Integers in decimal view ($\mathbb{Z}_{dub}$), continuation of Table 7 (and using $i^\star$ from Table 8)

3 Concluding remarks

When specifying a datatype of integers as an extension of the naturals, the unary view leads to satisfactory results. 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 the paper 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 in the specifications below.

3. The DDRS's for the binary view and the decimal view below are hardly intelligible unless one knows that the objective is to construct a commutative ring.

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.

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.

A survey of equational algebraic specifications for abstract data types is provided in [8]. In [4] one finds the general result that computable abstract data types 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 [6, 3, 7]. More recent applications of equational specifications can be found in [5].

References

[1] Bergstra, J.A. (2014). Four datatype defining rewrite systems for an abstract dataype 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. (2011). Inversive meadows and divisive meadows. *Journal of Applied Logic*, 9(3): 203–220.

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

[4] 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.

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

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

[7] 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.

[8] 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.

[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/