Meadow Enriched ACP Process Algebras

J.A. Bergstra
C.A. Middelburg

Report PRG0902 January 2009

J.A. Bergstra

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

tel. +31 20 525.7591
e-mail: janb@science.uva.nl

C.A. Middelburg

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

e-mail: kmiddelb@science.uva.nl

Programming Research Group Electronic Report Series

Meadow Enriched ACP Process Algebras

J.A. Bergstra and C.A. Middelburg

Programming Research Group, University of Amsterdam,
P.O. Box 41882, 1009 DB Amsterdam, the Netherlands
`J.A.Bergstra@uva.nl,C.A.Middelburg@uva.nl`

Abstract. We introduce the notion of an ACP process algebra. The models of the axiom system ACP are the origin of this notion. ACP process algebras have to do with processes in which no data are involved. We also introduce the notion of a meadow enriched ACP process algebra, which is a simple generalization of the notion of an ACP process algebra to processes in which data are involved. In meadow enriched ACP process algebras, the mathematical structure for data is a meadow.

Keywords: ACP process algebra, meadow enriched ACP process algebra.

1998 ACM Computing Classification: D.1.3, D.2.1, D.2.4, F.1.2, F.3.1.

1 Introduction

In many formalisms proposed for the description and analysis of processes in which data are involved, algebraic specifications of the data types concerned have to be given over and over again. This is also the case with the principal ACP-based formalisms proposed for the description and analysis of processes in which data are involved, to wit μCRL [11, 12] and PSF [13]. There is a mismatch between the process specification part and the data specification part of these formalisms. Firstly, there is a choice of one built-in type of processes, whereas there is a choice of all types of data that can be specified algebraically. Secondly, the semantics of the data specification part is its initial algebra in the case of PSF and its class of minimal Boolean preserving algebras in the case of μCRL, whereas the semantics of the process specification part is a model based on transition systems and bisimulation equivalence. Sticking to this mismatch, no stable axiomatizations in the style of ACP has emerged for process algebras that have to do with processes in which data is involved.

Our objective is to obtain a stable axiomatization in the style of ACP for process algebras that have to do with processes in which data is involved. To achieve this objective, we first introduce the notion of an ACP process algebra and then the notion of a meadow enriched ACP process algebra. ACP process algebras are essentially models of the axiom system ACP. Meadow enriched ACP process algebras are data enriched ACP process algebras in which the mathematical structure for data is a meadow. Meadows are defined for the first time in [7]. The prime example of meadows is the rational number field with the multiplicative inverse operation made total by imposing that the multiplicative inverse of zero is zero. Although the notion of a meadow enriched ACP process algebra is a simple generalization of the notion of an ACP process algebra, it is an interesting one: there is a multitude of finite and infinite meadows and meadows obviate the need for Boolean values and operations on data that yield Boolean values to deal with conditions on data.

In the work on ACP, the emphasis has always been on axiom systems. In this paper, we put the emphasis on algebras. That is, ACP process algebras are looked upon in the same way as groups, rings, fields, etc. are looked upon in universal algebra (see e.g. [9]). The set of equations that are taken to characterize ACP process algebras is a revision of the axiom system ACP. The revision is primarily a matter of streamlining. However, it also involves a minor generalization that allows for the generalization to meadow enriched ACP process algebras to proceed smoothly.

This paper is organized as follows. First, we give a brief summary of meadows (Section 2). Next, we introduce the notion of an ACP process algebra (Section 3) and the notion of an meadow enriched ACP process algebra (Section 4). Finally, we make some concluding remarks (Section 5).

2 Meadows

In the data enriched ACP process algebras introduced in this paper, the mathematical structure for data is a meadow. A meadow is a field with the multiplicative inverse operation made total by imposing that the multiplicative inverse of zero is zero. Meadows are defined for the first time in [7] and are investigated in e.g. [2, 5, 8]. In this section, we give a brief summary of meadows.

The signature of meadows is the same as the signature of fields. It is a one-sorted signature. We make the single sort explicit because we will extend this signature to a two-sorted signature in Section 4. The signature of meadows consists of the sort $\mathbf{Q}$ of *quantities* and the following constants and operators:

- the constants $0 : \mathbf{Q}$ and $1 : \mathbf{Q}$;
- the binary *addition* operator $+ : \mathbf{Q} \times \mathbf{Q} \to \mathbf{Q}$;
- the binary *multiplication* operator $\cdot : \mathbf{Q} \times \mathbf{Q} \to \mathbf{Q}$;
- the unary *additive inverse* operator $- : \mathbf{Q} \to \mathbf{Q}$;
- the unary *multiplicative inverse* operator ${}^{-1} : \mathbf{Q} \to \mathbf{Q}$.

We assume that there is a countably infinite set $\mathcal{U}$ of variables, which contains u, v and w, with and without subscripts. Terms are build as usual. We use infix notation for the binary operators $+$ and $\cdot$, prefix notation for the unary operator $-$, and postfix notation for the unary operator ${}^{-1}$. We use the usual precedence convention to reduce the need for parentheses. We introduce subtraction and division as abbreviations: $p - q$ abbreviates $p + (-q)$ and p/q abbreviates $p \cdot q^{-1}$. We freely use the numerals $2, 3, \ldots$.

The constants and operators from the signature of meadows are adopted from rational arithmetic, which gives an appropriate intuition about these constants and operators.

Table 1. Axioms for meadows

$(u + v) + w = u + (v + w)$	$(u \cdot v) \cdot w = u \cdot (v \cdot w)$	$(u^{-1})^{-1} = u$
$u + v = v + u$	$u \cdot v = v \cdot u$	$u \cdot (u \cdot u^{-1}) = u$
$u + 0 = u$	$u \cdot 1 = u$	
$u + (-u) = 0$	$u \cdot (v + w) = u \cdot v + u \cdot w$	

A meadow is an algebra with the signature of meadows that satisfies the equations given in Table 1. Thus, a meadow is a commutative ring with identity equipped with a multiplicative inverse operation ${}^{-1}$ satisfying the reflexivity equation $(u^{-1})^{-1} = u$ and the restricted inverse equation $u \cdot (u \cdot u^{-1}) = u$. From the equations given in Table 1, the equation $0^{-1} = 0$ is derivable.

A *non-trivial meadow* is a meadow that satisfies the *separation axiom*

$$0 \neq 1 \ .$$

A *cancellation meadow* is a meadow that satisfies the *cancellation axiom*

$$u \neq 0 \land u \cdot v = u \cdot w \Rightarrow v = w \ ,$$

or equivalently, the *general inverse law*

$$u \neq 0 \Rightarrow u \cdot u^{-1} = 1 \ .$$

In [7], cancellation meadows are called zero-totalized fields. An important property of cancellation meadows is the following: $0/0 = 0$, whereas $u/u = 1$ for $u \neq 0$.

3 ACP Process Algebras

In this section, we introduce the notion of an ACP process algebra. This notion originates from the models of the axiom system ACP, which was first presented in [3]. A comprehensive introduction to ACP can be found in [1, 10].

It is assumed that a fixed but arbitrary finite set A of *atomic action names*, with $\delta \notin \mathsf{A}$, has been given.

The signature of ACP process algebras is a one-sorted signature. We make the single sort explicit because we will extend this signature to a two-sorted signature in Section 4. The signature of ACP process algebras consists of the sort $\mathbf{P}$ of *processes* and the following constants, operators, and predicate symbols:

- the *deadlock* constant $\delta : \mathbf{P}$;
- for each $e \in \mathsf{A}$, the *atomic action* constant $e : \mathbf{P}$;
- the binary *alternative composition* operator $+ : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *sequential composition* operator $\cdot : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *parallel composition* operator $\parallel : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *left merge* operator $⌊\!⌊ : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;

– the binary *communication merge* operator $\mid : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
– for each $H \subseteq \mathsf{A}$, the unary *encapsulation* operator $\partial_H : \mathbf{P} \to \mathbf{P}$;
– the unary *atomic action* predicate symbol $\mathcal{A} : \mathbf{P}$.

We assume that there is a countably infinite set $\mathcal{X}$ of variables of sort $\mathbf{P}$, which contains x, y and z, with and without subscripts. Terms are built as usual. We use infix notation for the binary operators. We use the following precedence conventions to reduce the need for parentheses: the operator $+$ binds weaker than all other binary operators and the operator $\cdot$ binds stronger than all other binary operators. We write $\sum_{i \in \mathcal{I}} P_i$, where $\mathcal{I} = \{i_1, \ldots, i_n\}$ and $P_{i_1}, \ldots, P_{i_n}$ are terms of sort $\mathbf{P}$, for $P_{i_1} + \ldots + P_{i_n}$. The convention is that $\sum_{i \in \mathcal{I}} P_i$ stands for δ if $\mathcal{I} = \emptyset$.

Let P and Q be closed terms of sort $\mathbf{P}$. Intuitively, the constants and operators introduced above can be explained as follows:

– δ can neither perform an atomic action nor terminate successfully;
– e first performs atomic action e and then terminates successfully;
– $P + Q$ behaves either as P or as Q, but not both;
– $P \cdot Q$ first behaves as P and on successful termination of P it next behaves as Q;
– $P \parallel Q$ behaves as the process that proceeds with P and Q in parallel;
– $P \mathbin{⌊\!⌊} Q$ behaves the same as $P \parallel Q$, except that it starts with performing an atomic action of P;
– $P \mid Q$ behaves the same as $P \parallel Q$, except that it starts with performing an atomic action of P and an atomic action of Q synchronously;
– $\partial_H(P)$ behaves the same as P, except that atomic actions from H are blocked;
– $\mathcal{A}(P)$ holds if P is an atomic action.

The predicate $\mathcal{A}$ is a means to distinguish atomic actions from other processes. An alternate way to accomplish this is to have a subsort $\mathbf{A}$ of the sort $\mathbf{P}$. We have not chosen this alternate way because it complicates matters considerably. Moreover, we prefer to keep close to elementary algebraic specification (see e.g. [6]). By the notational convention introduced below, we seldom have to use the predicate $\mathcal{A}$ explicitly.

In equations between terms of sort $\mathbf{P}$, we will use a notational convention which requires the following assumption: there is a countably infinite set $\mathcal{X}' \subseteq \mathcal{X}$ that contains a, b and c, with and without subscripts, but does not contain x, y and z, with and without subscripts. Let ϕ be an equation between terms of sort $\mathbf{P}$, and let $\{a_1, \ldots, a_n\}$ be the set of all variables from $\mathcal{X}'$ that occur in ϕ. Then we write ϕ for $\mathcal{A}(x_1) \land \ldots \land \mathcal{A}(x_n) \Rightarrow \phi'$, where ϕ' is ϕ with, for all $i \in [1, n]$, all occurrences of a_i replaced by x_i, and $x_1, \ldots, x_n$ are variables from $\mathcal{X}$ that do not occur in ϕ.

An ACP process algebra is an algebra with the signature of ACP process algebras that satisfies the formulas given in Table 2. Three equations in this table are actually schemas of equations: e is a syntactic variable which stands for an arbitrary constant of sort $\mathbf{P}$. A side condition is added to two schemas

Table 2. Axioms for ACP process algebras

$x + y = y + x$		$x \parallel y = (x \mathbin{⌊\!⌊} y + y \mathbin{⌊\!⌊} x) + x \mid y$
$(x + y) + z = x + (y + z)$		$a \mathbin{⌊\!⌊} x = a \cdot x$
$x + x = x$		$a \cdot x \mathbin{⌊\!⌊} y = a \cdot (x \parallel y)$
$(x + y) \cdot z = x \cdot z + y \cdot z$		$(x + y) \mathbin{⌊\!⌊} z = x \mathbin{⌊\!⌊} z + y \mathbin{⌊\!⌊} z$
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$		$a \mid b \cdot x = (a \mid b) \cdot x$
$x + \delta = x$		$a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$
$\delta \cdot x = \delta$		$(x + y) \mid z = x \mid z + y \mid z$
		$x \mid y = y \mid x$
		$(x \mid y) \mid z = x \mid (y \mid z)$
$\partial_H(e) = e$	if $e \notin H$	$\delta \mid x = \delta$
$\partial_H(e) = \delta$	if $e \in H$	
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$		$\mathcal{A}(e)$
$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$		$\mathcal{A}(x) \wedge \mathcal{A}(y) \Rightarrow \mathcal{A}(x \mid y)$

to restrict the constants for which the syntactic variable stands. The number of proper equations is still finite because there exists only a finite number of constant of sort **P**.

Because the notational convention introduced above is used, the four equations in Table 2 that are actually conditional equations look the same as their counterpart in the axiom system ACP. It happens that these conditional equations allow for the generalization to meadow enriched ACP process algebras to proceed smoothly. Apart from this, the set of formulas given in Table 2 differs from the axiom system ACP on three points. Firstly, the equations $x \mid y = y \mid x$, $(x \mid y) \mid z = x \mid (y \mid z)$, and $\delta \mid x = \delta$ have been added. In the axiom system ACP, all closed substitution instances of these equations are derivable. Secondly, the equations $a \cdot x \mid b = (a \mid b) \cdot x$ and $x \mid (y + z) = x \mid y + x \mid z$ have been removed. These equations can be derived using the added equation $x \mid y = y \mid x$. Thirdly, the formulas $\mathcal{A}(e)$ and $\mathcal{A}(x) \wedge \mathcal{A}(y) \Rightarrow \mathcal{A}(x \mid y)$ have been added. They express that the processes denoted by constants of sort **P** are atomic actions and that the processes that result from the communication merge of two atomic actions are atomic actions. This does not exclude that there are additional atomic actions, which is impossible in the case of ACP.

Not all processes in a ACP process algebra have to be interpretations of closed terms, even if all atomic actions are interpretations of closed terms. The processes concerned may be solutions of sets of recursion equations. It is customary to restrict the attention to ACP process algebras satisfying additional axioms by which sets of recursion equations that fulfil a guardedness condition have unique solutions. For an comprehensive treatment of this issue, the reader is referred to [1].

4 Meadow Enriched ACP Process Algebras

In this section, we introduce the notion of an meadow enriched ACP process algebra. This notion is a simple generalization of the notion of an ACP process algebra introduced in Section 3 to processes in which data are involved. The elements of a meadow are taken as data.

The signature of meadow enriched ACP process algebras is a two-sorted signature. It consists of the sorts, constants and operators from the signatures of ACP process algebras and meadows and in addition the following operators:

- for each $n \in \mathbb{N}$ and $e \in \mathsf{A}$, the n-ary *data handling atomic action* operator $e : \underbrace{\mathbf{Q} \times \cdots \times \mathbf{Q}}_{n \text{ times}} \to \mathbf{P}$;
- the binary *guarded command* operator $:\to\, : \mathbf{Q} \times \mathbf{P} \to \mathbf{P}$.

We take the variables in $\mathcal{U}$ for the variables of sort $\mathbf{Q}$. We assume that the sets $\mathcal{U}$ and $\mathcal{X}$ are disjunct. Terms are built as usual for a many-sorted signature (see e.g. [14, 15]). We use the same notational conventions as before. In addition, we use infix notation for the binary operator $:\to$.

Let $p_1, \ldots, p_n$ and p be closed terms of sort $\mathbf{Q}$ and P be a closed term of sort $\mathbf{P}$. Intuitively, the additional operators can be explained as follows:

- $e(p_1, \ldots, p_n)$ first performs data handling atomic action $e(p_1, \ldots, p_n)$ and then terminates successfully;
- $p :\to P$ behaves as the process P under the condition that $p = 0$ holds.

The different guarded command operators that have been proposed before in the setting of ACP have one thing in common: their first operand is considered to stand for an element of the domain of a Boolean algebra (see e.g. [4]). In contrast with those guarded command operators, the first operand of the guarded command operator introduced here is considered to stand for an element of the domain of a meadow.

A meadow enriched ACP process algebra is an algebra with the signature of meadow enriched ACP process algebras that satisfies the formulas given in Tables 2 and 3. Like in Table 2, some formulas in Table 3 are actually schemas of formulas: e and e' are syntactic variables which stand for arbitrary constants of sort $\mathbf{P}$ and, in addition, n and m stand for arbitrary natural numbers.

For meadow enriched ACP process algebras that satisfy the cancellation axiom, the first five equations concerning the guarded command operator can easily be understood by taking the view that 0 and 1 represent the Boolean values T and F, respectively. In that case, we have that

- p/p models the test that yields T if $p = 0$ and F otherwise;
- if both p and q are equal to 0 or 1, then $1 - p$ models $\neg p$, $p \cdot q$ models $p \vee q$, and consequently $1 - (1 - p) \cdot (1 - q)$ models $p \wedge q$.

From this view, the equations given in the upper half of Table 3 differ from the axioms for the most general kind of guarded command operator that has

Table 3. Additional axioms for meadow enriched ACP process algebras

$0 :\rightarrow x = x$	$u :\rightarrow \delta = \delta$
$1 :\rightarrow x = \delta$	$u :\rightarrow (x + y) = u :\rightarrow x + u :\rightarrow y$
$u :\rightarrow x = u/u :\rightarrow x$	$u :\rightarrow x \cdot y = (u :\rightarrow x) \cdot y$
$u :\rightarrow (v :\rightarrow x) = (1 - (1 - u/u) \cdot (1 - v/v)) :\rightarrow x$	$(u :\rightarrow x) \mathbin{\lfloor\!\lfloor} y = u :\rightarrow (x \mathbin{\lfloor\!\lfloor} y)$
$u :\rightarrow x + v :\rightarrow x = (u/u \cdot v/v) :\rightarrow x$	$(u :\rightarrow x) \mid y = u :\rightarrow (x \mid y)$
	$\partial_H(u :\rightarrow x) = u :\rightarrow \partial_H(x)$
$e(u_1, \dots, u_n) \mid e'(v_1, \dots, v_n) =$ $(u_1 - v_1) :\rightarrow (\dots :\rightarrow ((u_n - v_n) :\rightarrow (e \mid e')(u_1, \dots, u_n)) \cdots)$	
$e(u_1, \dots, u_n) \mid e'(v_1, \dots, v_m) = \delta$	if $n \neq m$
$\partial_H(e(u_1, \dots, u_n)) = e(u_1, \dots, u_n)$	if $e \notin H$
$\partial_H(e(u_1, \dots, u_n)) = \delta$	if $e \in H$
$\mathcal{A}(e(u_1, \dots, u_n))$	

been proposed in the setting of ACP (see e.g. [4]) on two points only. Firstly, the equation $u :\rightarrow x = u/u :\rightarrow x$ has been added. This equation formalizes the informal explanation of the guarded command given above. Secondly, the equation $x \mid (u :\rightarrow y) = u :\rightarrow (x \mid y)$ has removed. This equation can be derived using the equation $x \mid y = y \mid x$ from Table 2.

The equations in Table 3 concerning the communication merge of data handling atomic actions formalize the intuition that two data handling atomic actions $e(p_1, \dots, p_n)$ and $e'(q_1, \dots, q_m)$ cannot be performed synchronously if $n \neq m$ or $p_1 \neq q_1$ or ... or $p_n \neq q_n$. The equations concerning the encapsulation of data handling atomic actions formalize the way in which it is dealt with in μCRL and PSF. The formula concerning the atomic action predicate simply expresses that data handling atomic actions are also atomic actions.

5 Conclusions

We have introduced the notion of an ACP process algebra. The set of equations that have been taken to characterize ACP process algebras is a revision of the axiom system ACP. We consider this revision worth mentioning of itself. We have also introduced the notion of a meadow enriched ACP process algebra. This notion is a simple generalization of the notion of an ACP process algebra to processes in which data are involved, the mathematical structure of data being a meadow. The primary mathematical structure for calculations is unquestionably a field, and a meadow differs from a field only in that the multiplicative inverse operation is made total by imposing that the multiplicative inverse of zero is zero. Therefore, we consider a combination of ACP process algebras and meadows like the one made in this paper, a combination with potentially many applications.

References

1. Baeten, J.C.M., Weijland, W.P.: Process Algebra, *Cambridge Tracts in Theoretical Computer Science*, vol. 18. Cambridge University Press, Cambridge (1990)
2. Bergstra, J.A., Hirschfeld, Y., Tucker, J.V.: Meadows and the equational specification of division. `arXiv:0901.0823v1 [math.RA]` at `http://arxiv.org/` (2009)
3. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Information and Control **60**(1/3), 109–137 (1984)
4. Bergstra, J.A., Middelburg, C.A.: Splitting bisimulations and retrospective conditions. Information and Computation **204**(7), 1083–1138 (2006)
5. Bergstra, J.A., Ponse, A.: A generic basis theorem for cancellation meadows. `arXiv:0803.3969v2 [math.RA]` at `http://arxiv.org/` (2008)
6. Bergstra, J.A., Tucker, J.V.: Elementary algebraic specifications of the rational complex numbers. In: K. Futatsugi, et al. (eds.) Goguen Festschrift, *Lecture Notes in Computer Science*, vol. 4060, pp. 459–475. Springer-Verlag (2006)
7. Bergstra, J.A., Tucker, J.V.: The rational numbers as an abstract data type. Journal of the ACM **54**(2), Article 7 (2007)
8. Bethke, I., Rodenburg, P.H.: Some properties of finite meadows. `arXiv:0712.0917v1 [math.RA]` at `http://arxiv.org/` (2007)
9. Burris, S., Sankappanavar, H.P.: A Course in Universal Algebra, *Graduate Texts in Mathematics*, vol. 78. Springer-Verlag, Berlin (1981)
10. Fokkink, W.J.: Introduction to Process Algebra. Texts in Theoretical Computer Science, An EATCS Series. Springer-Verlag, Berlin (2000)
11. Groote, J.F., Ponse, A.: Proof theory for μCRL: A language for processes with data. In: D.J. Andrews, J.F. Groote, C.A. Middelburg (eds.) Semantics of Specification Languages, Workshops in Computing Series, pp. 232–251. Springer-Verlag (1994)
12. Groote, J.F., Ponse, A.: The syntax and semantics of μCRL. In: A. Ponse, C. Verhoef, S.F.M. van Vlijmen (eds.) Algebra of Communicating Processes 1994, Workshops in Computing Series, pp. 26–62. Springer-Verlag (1995)
13. Mauw, S., Veltink, G.J.: A process specification formalism. Fundamenta Informaticae **13**(2), 85–139 (1990)
14. Sannella, D., Tarlecki, A.: Algebraic preliminaries. In: E. Astesiano, H.J. Kreowski, B. Krieg-Brückner (eds.) Algebraic Foundations of Systems Specification, pp. 13–30. Springer-Verlag, Berlin (1999)
15. Wirsing, M.: Algebraic specification. In: J. van Leeuwen (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 675–788. Elsevier, Amsterdam (1990)

[PRG0708] B. Diertens, *Software (Re-)Engineering with PSF III: an IDE for PSF*, Programming Research Group - University of Amsterdam, 2007.

[PRG0707] J.A. Bergstra and C.A. Middelburg, *An Interface Group for Process Components,* Programming Research Group - University of Amsterdam, 2007.

[PRG0706] J.A. Bergstra, Y. Hirschfeld, and J.V. Tucker, *Skew Meadows,* Programming Research Group - University of Amsterdam, 2007.

[PRG0705] J.A. Bergstra, Y. Hirschfeld, and J.V. Tucker, *Meadows*, Programming Research Group - University of Amsterdam, 2007.

[PRG0704] J.A. Bergstra and C.A. Middelburg, *Machine Structure Oriented Control Code Logic (Extended Version),* Programming Research Group - University of Amsterdam, 2007.

[PRG0703] J.A. Bergstra and C.A. Middelburg, *On the Operating Unit Size of Load/Store Architectures,* Programming Research Group - University of Amsterdam, 2007.

[PRG0702] J.A. Bergstra and A. Ponse, *Interface Groups and Financial Transfer Architectures,* Programming Research Group - University of Amsterdam, 2007.

[PRG0701] J.A. Bergstra, I. Bethke, and M. Burgess, *A Process Algebra Based Framework for Promise Theory*, Programming Research Group - University of Amsterdam, 2007.

[PRG0610] J.A. Bergstra and C.A. Middelburg, *Parallel Processes with Implicit Computational Capital,* Programming Research Group - University of Amsterdam, 2006.

[PRG0609] B. Diertens, *Software (Re-)Engineering with PSF II: from architecture to implementation*, Programming Research Group - University of Amsterdam, 2006.

[PRG0608] A. Ponse and M.B. van der Zwaag, *Risk Assessment for One-Counter Threads*, Programming Research Group - University of Amsterdam, 2006.

[PRG0607] J.A. Bergstra and C.A. Middelburg, *Synchronous Cooperation for Explicit Multi-Threading,* Programming Research Group - University of Amsterdam, 2006.

[PRG0606] J.A. Bergstra and M. Burgess, *Local and Global Trust Based on the Concept of Promises*, Programming Research Group - University of Amsterdam, 2006.

[PRG0605] J.A. Bergstra and J.V. Tucker, *Division Safe Calculation in Totalised Fields,* Programming Research Group - University of Amsterdam, 2006.

[PRG0604] J.A. Bergstra and A. Ponse, *Projection Semantics for Rigid Loops,* Programming Research Group - University of Amsterdam, 2006.

[PRG0603] J.A. Bergstra and I. Bethke, *Predictable and Reliable Program Code: Virtual Machine-based Projection Semantics (submitted for inclusion in the Handbook of Network and Systems Administration),* Programming Research Group - University of Amsterdam, 2006.

[PRG0602] J.A. Bergstra and A. Ponse, *Program Algebra with Repeat Instruction,* Programming Research Group - University of Amsterdam, 2006.

[PRG0601] J.A. Bergstra and A. Ponse, *Interface Groups for Analytic Execution Architectures,* Programming Research Group - University of Amsterdam, 2006.

[PRG0505] B. Diertens, *Software (Re-)Engineering with PSF,* Programming Research Group - University of Amsterdam, 2005.

[PRG0504] P.H. Rodenburg, *Piecewise Initial Algebra Semantics,* Programming Research Group - University of Amsterdam, 2005.

[PRG0503] T.D. Vu, *Metric Denotational Semantics for BPPA,* Programming Research Group - University of Amsterdam, 2005.

[PRG0502] J.A. Bergstra, I. Bethke, and A. Ponse, *Decision Problems for Pushdown Threads,* Programming Research Group - University of Amsterdam, 2005.