Process algebra with explicit termination

Eindhoven University of Technology
Department of Mathematics and Computing Science

Process Algebra with Explicit Termination

by

J.C.M. Baeten

00/02

ISSN 0926-4515

editors: prof.dr. J.C.M. Baeten
prof.dr. P.A.J. Hilbers

Reports are available at:
http://www.win.tue.nl/win/cs

Computing Science Reports 00/02
Eindhoven, February 2000

Process Algebra with Explicit Termination

J.C.M. Baeten
Department of Mathematics and Computing Science,
Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands,
josb@win.tue.nl, http://www.win.tue.nl/win/cs/fm/josb/

Abstract

In ACP-style process algebra, the interpretation of a constant atomic action combines action execution with termination. In a setting with timing, different forms of termination can be distinguished: some time termination, current time slice termination, urgent termination, termination in a virtual state. In a setting with the silent action τ, we also have silent termination.

This leads to problems with the interpretation of atomic actions in timed theories that involve some form of the empty process or some form of the silent action.

Reflection on these problems lead to a re-design of basic process algebra, where action execution and termination are separated. Instead of actions as constants, we have action prefix operators. Sequential composition remains a basic operator, and thus we have two basic constants for termination, δ for unsuccessful termination (deadlock) and ϵ for successful termination (skip). Standard BPA, PA, ACP become SRM specifications of the new approach. The new approach has definite advantages over the standard approach.

1 Introduction

In ACP-style process algebra (see e.g. [11, 8, 7]), the interpretation of a constant atomic action combines action execution with termination. In a setting with timing, different forms of termination can be distinguished: some time termination, current time slice termination, urgent termination, termination in a virtual state. In a setting with the silent action τ, we also have silent termination.

This leads to problems with the interpretation of atomic actions in timed theories that involve some form of the empty process or some form of the silent action, see [3, 6].

Reflection on these problems lead to a re-design of basic process algebra, where action execution and termination are separated. Instead of actions as constants, we have action prefix operators as in CCS [18] or in CSP [17]. As in CSP, but different from CCS, we have that sequential composition remains a basic operator. As a consequence, we have two basic constants for termination, δ for unsuccessful termination (deadlock) and ϵ for successful termination (skip). We still have the advantage of ACP over CCS, CSP that we have a strictly algebraical approach: we start out from a set of axioms, and can consider several semantical models.

Standard BPA, PA, ACP become SRM specifications of the new approach. The new approach has definite advantages over the standard approach.

We have better separation of action execution and termination, and moreover better separation of atomic actions as a parameter of the theory and as signature elements. We can define a minimal process algebra without sequential composition, and this makes formulation of concepts such as structural induction, linearity, elimination and guardedness easier. The difference between the silent step τ and the empty step ϵ becomes clearer.

In the operational semantics, we have no need for separate terminating action executions. We have a natural restriction of iteration to action prefix iteration, allowing complete axiomatizations in more cases. Moreover, prefix iteration is sufficient to formulate time iteration and embedding of untimed into timed theories, and sufficient to formulate divergent behaviour and fair iteration.

1.1 Acknowledgements

The author wishes to acknowledge suggestions and helpful remarks by Jan Bergstra and Alban Ponse (Univ. of Amsterdam), Leszek Holenderski, Kees Middelburg, Michel Reniers, Jeroen Voeten and Marc Voorhoeve (all Eindhoven University of Technology).

2 Sequential Process Algebra

We start out from the process algebra SPA, *Sequential Process Algebra* (or: algebra of sequential processes). SPA is a modification of the basic process algebra BPA, where action constants are replaced by action prefix operators. We assume we have given a set of actions A. This set, usually finite, is considered a parameter of the theory. The signature elements are:

- Binary operator $+$ denotes *alternative composition* or choice. Process $x + y$ executes either x or y, but not both. The choice is resolved upon execution of the first action.
- Binary operator $\cdot$ denotes *sequential composition*. We choose to have sequential composition as a basic operator. This is different from CCS (see [18]) where sequential composition is a derived operator. In our view, this is in disregard to the central role of sequential composition in all specification and programming languages. As a result, we can distinguish between successful and unsuccessful termination (deadlock).
- Constant δ denotes *inaction* (or deadlock), and is the neutral element of alternative composition. Process δ cannot execute any action, and cannot terminate.
- Constant ϵ denotes the *empty process* or *skip*. It is the neutral element of sequential composition. Process ϵ cannot execute any action, but terminates successfully.
- We have a unary operator $a.$ for each $a \in A$ called *action prefix*. Process $a.x$, usually written ax, executes action a and then proceeds as x. Putting a constant for x, we have the basic processes $a\delta$ (deadlock upon execution of a) and $a\epsilon$ (successful termination upon execution of a).

The process algebra SPA is axiomatised by axioms A1-10 in Table 1. Axiom A10 is actually an axiom scheme: we have such an axiom for each action $a \in A$. Prefix operators always bind stronger than other operators, $+$ always binds weaker.

$x+y=y+x$	A1	$x+\delta=x$	A6
$(x+y)+z=x+(y+z)$	A2	$\delta\cdot x=\delta$	A7
$x+x=x$	A3	$\epsilon\cdot x=x$	A8
$(x+y)\cdot z=x\cdot z+y\cdot z$	A4	$x\cdot\epsilon=x$	A9
$(x\cdot y)\cdot z=x\cdot(y\cdot z)$	A5	$ax\cdot y=a(x\cdot y)$	A10

Table 1: Axioms of SPA ($a\in A$).

Proposition 1 For all closed SPA-terms, axiom A5 is derivable from the other axioms of SPA.

Proof In order to prove $(x\cdot y)\cdot z=x\cdot(y\cdot z)$ for all closed SPA-terms x,y,z, we use structural induction on term x. When we prove the equation for a certain triple x,y,z, we can use as the induction hypothesis that for all subterms x' of x, and for all terms v,w, we have $(x'\cdot v)\cdot w=x'\cdot(v\cdot w)$. Now we have the following case distinction.

1. $x=\delta$. Then $(\delta\cdot y)\cdot z=\delta\cdot z=\delta=\delta\cdot(y\cdot z)$.
2. $x=\epsilon$. Then $(\epsilon\cdot y)\cdot z=y\cdot z=\epsilon\cdot(y\cdot z)$.
3. x has the form of an action prefix, say $x=ax'$ for certain $a\in A$ and subterm x'. Then $(ax'\cdot y)\cdot z=a(x'\cdot y)\cdot z=a((x'\cdot y)\cdot z)=a(x'\cdot(y\cdot z))=ax'\cdot(y\cdot z)$.
4. x has the form of a sum, say $x=x'+x''$ for certain subterms x',x''. Then $((x'+x'')\cdot y)\cdot z=(x'\cdot y+x''\cdot y)\cdot z=(x'\cdot y)\cdot z+(x''\cdot y)\cdot z=x'\cdot(y\cdot z)+x''\cdot(y\cdot z)=(x'+x'')\cdot(y\cdot z)$.
5. x has the form of a product, say $x=x'\cdot x''$ for certain subterms x',x''. Then $((x'\cdot x'')\cdot y)\cdot z=(x'\cdot(x''\cdot y))\cdot z=x'\cdot((x''\cdot y)\cdot z)=x'\cdot(x''\cdot(y\cdot z))=(x'\cdot x'')\cdot(y\cdot z)$. □

Thus, associativity of sequential composition (A5) can be proved on the initial algebra model of the theory. This could be a reason not to include this axiom in our theory. We do so, nevertheless, since it is such a basic result, that we will always assume that it holds for all processes. Its status is comparable to the axioms of standard concurrency of [12].

Since we have action prefixing as a separate operator, besides sequential composition, it becomes possible to consider the subalgebra that arises if we delete the sequential composition operator. The signature of MPA (this stands for Minimal Process Algebra, this acronym was introduced in [14]) is the signature of SPA without sequential composition. The process algebra MPA consists of the axioms A1,2,3,6 of Table 1. Using these axioms, each closed MPA-term t can be written in one of the following two forms:

1. MPA $\vdash t=\delta+a_1t_1+\cdots+a_nt_n$, or
2. MPA $\vdash t=\epsilon+a_1t_1+\cdots+a_nt_n$,

for certain $n\in\mathbb{N}$, $a_i\in A$ and (smaller) MPA-terms t_i ($i\leq n$). (In the first case, we can omit δ when $n>0$.)

The next proposition shows that each closed SPA-term can be reduced to a closed MPA-term, so sequential composition can be eliminated from closed terms. This is the so-called *elimination theorem*. This allows the use of structural induction in proofs and axiomatisations without considering the case of sequential composition. The separation of action prefixing and sequential composition allows easier formulations of the elimination theorem and structural induction.

Being able to reduce each closed term to a term without sequential composition does not imply that sequential composition is not important. In fact, when we add recursion we can define only regular processes just using prefixing, but can define non-regular processes by the use of sequential composition.

Proposition 2 Let t be a closed SPA-term. Then there is a closed MPA-term s such that $\mathrm{SPA} \vdash t = s$.

Proof We can turn the axioms A3-10 of SPA into rewrite rules, by orienting them from left to right. We obtain a confluent and terminating term rewrite system modulo A1-2. We find s by reducing t to normal form. This proof is like several examples in [8] or [7]. □

This proof shows more than the statement of the theorem. From it also follows the fact, that SPA is a conservative extension of MPA.

Next, we provide a model for SPA on the basis of structured operational rules (so-called *SOS rules*) in the style of Plotkin (see [19]). The rules in Table 2 define the following relations on closed SPA-terms: binary relations $. \xrightarrow{a} .$ (for $a \in A$) and a unary relation $\downarrow$. Intuitively, they have the following meaning:

- $x \xrightarrow{a} x'$ means that x evolves into x' by executing atomic action a;
- $x \downarrow$ means that x has an option to terminate successfully (without executing an action)

Thus, the relations concern action execution and termination, respectively, we do not have need of a mixed relation $. \xrightarrow{a} \surd$ as in [8] or [7].

$$\epsilon \downarrow \qquad\qquad ax \xrightarrow{a} x$$

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \qquad \frac{x \downarrow}{x + y \downarrow} \qquad \frac{y \downarrow}{x + y \downarrow}$$

$$\frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y} \qquad \frac{x \downarrow, y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'} \qquad \frac{x \downarrow, y \downarrow}{x \cdot y \downarrow}$$

Table 2: Deduction rules for SPA ($a \in A$).

The rules provide a transition system for each closed term. The rules for MPA are given by the first two lines of Table 2. Next, we define an equivalence relation on the resulting transition systems in the standard way.

Definition 3 Let R be a binary *symmetric* relation on closed terms. We say R is a *bisimulation* if the following holds:

- whenever $R(x,y)$ and $x \xrightarrow{a} x'$ then there is a term y' such that $y \xrightarrow{a} y'$ and $R(x',y')$
- whenever $R(x,y)$ and $x \downarrow$ then $y \downarrow$

We say two closed terms t, s are *bisimulation equivalent* or *bisimilar*, notation $t \leftrightarroweq s$ if there is a bisimulation R with $R(s,t)$.

Proposition 4 Bisimulation equivalence is a congruence relation on closed SPA-terms.

Proof This is a standard result following from the format of the deduction rules, see e.g. [7]. □

Theorem 5 *The theory SPA is sound and complete for the model of transition systems modulo bisimulation, i.e. for all closed terms t, s we have*

$$SPA \vdash t = s \iff t \leftrightarroweq s$$

Proof This is also a standard result, based on the elimination theorem. We can follow the recipe of [7]. □

We can get back the original formulation of the process algebra of [11, 8, 7] as follows. We define new constants a by the equation $a = a\epsilon$, for each $a \in A$. Then, we reduce the signature by deleting the prefix operators. The subalgebra of the initial algebra that is obtained by this reduced signature is now completely axiomatised by the theory $\mathrm{BPA}_{\delta\epsilon}$ of [8, 7]. Following [2], we call $\mathrm{BPA}_{\delta\epsilon}$ an SRM-specification (Subalgebra of Reduced Model Specification) of SPA.

Then, we can reduce further by deleting ϵ, or also δ, and obtain the SRM specifications BPA_{δ} resp. BPA of [11, 8, 7]. This means that all specifications and verifications of systems that have been obtained in the last decades remain valid.

3 Renaming and Encapsulation

The theory can now be further developed as in the standard case. As an example, we consider the extension with renaming operators and blocking operators.

First, we consider renaming operators. For each unary function f on A, we have a unary renaming operator ρ_f, that renames each action a into $f(a)$, and does nothing else. The axiomatization in Table 3 follows the structure of SPA-terms, the deduction rules are given in Table 4. Since we have a clear separation between the set of atomic actions A and the action prefix operators, we also have a clear separation between the renaming *function* f (given as a parameter) and the renaming *operator* ρ_f (in the signature of the algebra). Further, we have no confusion between the constant δ and the atomic actions. This simplifies the deduction rules.

Based on the format of the deduction rules, we can prove that the extension with renaming operators is a conservative extension of SPA and that the axiomatisation is sound and complete for the model of transition systems modulo bisimulation. Notice that the axiom

$\rho_f(\delta) = \delta$	$\rho_f(x \cdot y) = \rho_f(x) \cdot \rho_f(y)$
$\rho_f(\epsilon) = \epsilon$	$\rho_f(x + y) = \rho_f(x) + \rho_f(y)$
$\rho_f(ax) = f(a)\rho_f(x)$	

Table 3: Axioms of renaming operators ($a \in A, f : A \to A$).

$$\frac{x \xrightarrow{a} x'}{\rho_f(x) \xrightarrow{f(a)} \rho_f(x')} \qquad \frac{x \downarrow}{\rho_f(x) \downarrow}$$

Table 4: Deduction rules for renaming ($a \in A, f : A \to A$).

$\rho_f(x \cdot y) = \rho_f(x) \cdot \rho_f(y)$ is derivable from the other axioms for all closed terms (using structural induction on x).

The renaming operators are clearly distinguished from the blocking or encapsulation operator ∂_H that prohibits execution of actions from H, a subset of A. Axiomatization and deduction rules follow the same pattern, nonetheless, see Tables 5 and 6, and the same results concerning conservativity and completeness can be established.

$\partial_H(\delta) = \delta$	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$
$\partial_H(\epsilon) = \epsilon$	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$
$\partial_H(ax) = a\partial_H(x)$ if $a \notin H$	
$\partial_H(ax) = \delta$ if $a \in H$	

Table 5: Axioms of encapsulation operators ($a \in A, H \subseteq A$).

4 Infinite Processes

4.1 Iteration

An interesting extension is the extension with the iteration prefix. For each action $a \in A$, we have a new unary operator a^* called *iteration prefix*. The process a^*x can execute a a number of times before starting the execution of x. Note that this construction allows unbounded behaviour, for instance $a^*\delta$ will execute a an unbounded number of times, i.e. infinitely often.

The process algebra SPA extended with prefix iteration is called SPA*, and axiomatized by the axioms in Tables 1 and 7, MPA with prefix iteration is called MPA*, and is axiomatized by axioms A1,2,3,6 in Table 1 and the two axioms on the left hand side in Table 7.

Proposition 6 The theory MPA* is a sound and complete axiomatization of the model of transition systems modulo bisimulation.

$$\frac{x \xrightarrow{a} x',\ a \notin H}{\partial_H(x) \xrightarrow{a} \partial_H(x')} \qquad \frac{x \downarrow}{\partial_H(x) \downarrow}$$

Table 6: Deduction rules for encapsulation ($a \in A, H \subseteq A$).

$$a(a^*x) + x = a^*x \qquad a^*(x \cdot y) = a^*x \cdot y$$
$$a^*(a^*x) = a^*x$$

Table 7: Axioms of iteration prefix ($a \in A$).

Proof This is a straightforward adaptation of the proof of Fokkink in [14]. □

Proposition 7 The theory SPA* is a sound and complete axiomatization of the model of transition systems modulo bisimulation.

Proof First, we extend the elimination theorem, showing that each closed SPA*-term can be reduced to a closed MPA*-term. Then, we show that all additional axioms are sound, and invoke the previous proposition. □

Note that associativity of sequential composition still follows from the other axioms for all closed SPA*-terms (add a case in the induction that goes

$$(a^*x \cdot y) \cdot z = (a^*(x \cdot y)) \cdot z = a^*((x \cdot y) \cdot z) = a^*(x \cdot (y \cdot z)) = a^*x \cdot (y \cdot z)).$$

Note that here we see an advantage of just having prefix iteration. If we add iteration as a binary operator on SPA, where process x^*y can iterate the behaviour of x until exiting by doing y, then a complete finite axiomatization cannot be found. This was shown by Sewell [20]. On the other hand, there are extensions of action prefix iteration that still have finite axiomatizations, see e.g. [1, 15].

We can look at the combination of prefix iteration and renaming. Here, the axioms we have are definitely not complete, we need the following extra axiom:

$$\rho_f(a^*x) = f(a)^*\rho_f(x).$$

Using this axiom, the renaming operator can be eliminated from all closed terms, and previous results go through.

$$a^*x \xrightarrow{a} a^*x \qquad \frac{x \xrightarrow{b} x'}{a^*x \xrightarrow{b} x'} \qquad \frac{x \downarrow}{a^*x \downarrow}$$

Table 8: Deduction rules for iteration prefix ($a, b \in A$).

4.2 Recursion

The first axiom of prefix iteration is a specific instance of a *recursive equation*. It is standard to use recursive equations to specify processes with possible infinite behaviour, see e.g. [8] or [7]. We proceed to define recursion in our setting.

Let V be a set of variables ranging over processes. A *recursive specification* $E = E(V)$ is a set of equations $E = \{X = t_X \mid X \in V\}$ where each t_X is a term over the signature in question (for instance, SPA) and variables from V. A *solution* of a recursive specification $E(V)$ in our theory is a set of processes $\{\langle X|E\rangle \mid X \in V\}$ in some model of the theory such that the equations of $E(V)$ hold, if for all $X \in V$, X stands for $\langle X|E\rangle$. Mostly, we are interested in one particular variable $X \in V$.

Let t be a term containing a variable X. We call an occurrence of X in t *guarded* if this occurrence of X is in the scope of an action prefix operator (i.e., t has a subterm of the form as, and this X occurs in s). This is an easier formulation of guardedness than in the standard case.

We call a recursive specification *guarded* if all occurrences of all its variables in the right-hand sides of all its equations are guarded or it can be rewritten to such a recursive specification using the axioms of the theory and the equations of the specification.

We can formulate the following principles:

- RDP (the *Recursive Definition Principle*): each recursive specification has at least one solution
- RSP (the *Recursive Specification Principle*): each *guarded* recursive specification has at most one solution

Different models of SPA will satisfy none, one or both of these principles. Let us look at the transition system models in particular.

Consider a recursive specification E. For each variable X of E, we can add a new constant $\langle X|E\rangle$ to our syntax. Table 9 provides deduction rules for these constants. They come down to looking upon $\langle X|E\rangle$ as the process $\langle t_X|E\rangle$, which is t_X with, for all $Y \in V$, all occurrences of Y in t_X replaced by $\langle Y|E\rangle$.

$$\frac{\langle t_X|E\rangle \xrightarrow{a} y}{\langle X|E\rangle \xrightarrow{a} y} \qquad \frac{\langle t_X|E\rangle \downarrow}{\langle X|E\rangle \downarrow}$$

Table 9: Deduction rules for recursion ($a \in A$).

Now if we add such constants $\langle X|E\rangle$ for all specifications E to our syntax, we obtain a model $\mathbb{G}^\infty$ for SPA that satisfies RDP and RSP (see e.g. [8]). If we add constants $\langle X|E\rangle$ only for guarded E, we obtain a model $\mathbb{G}$ satisfying RSP. The model $\mathbb{G}$ is really smaller than $\mathbb{G}^\infty$, since using unguarded recursion we can specify infinitely branching processes, whereas for guarded recursion we can always get a finitely branching solution. Thus, RDP doesn't hold any more on the second model. Note that, due to the presence of the constant ϵ, a difference between SPA and the standard theory BPA is that finite guarded recursion allows the specification of a process with unbounded branching (see [13]).

The third possibility is adding still fewer constants, adding only $\langle X|E\rangle$ over the syntax of MPA. We call an equation *linear* if it is guarded and just uses the signature of MPA. Notice this is a very concise definition of linearity. Every linear specification is equivalent to one where all equations have one of the two forms

1. $X = \delta + a_1X_1 + \cdots + a_nX_n$, or
2. $X = \epsilon + a_1X_1 + \cdots + a_nX_n$,

for certain $n \in \mathbb{N}, a_i \in A, X_i \in V$.

The model $\mathbf{R}$ of SPA is obtained if we add constants $\langle X|E\rangle$ only for finite linear E. $\mathbf{R}$ is the model of *regular* processes, it is equivalent to the model of finite transition systems modulo bisimulation, see e.g. [10]. Again we can establish that $\mathbf{R}$ is really smaller than $\mathbb{G}$, a process in the difference is the counter C defined by the following specification (p standing for plus, m for minus):

$$C = T \cdot C \qquad T = pS \qquad S = m\epsilon + T \cdot S.$$

There is no finite guarded specification over MPA that has the counter as a solution, as the counter has infinitely many different states. Thus, we see sequential composition does add expressive power.

Finally, the term model $\mathbf{P}$ of SPA* is even smaller than $\mathbf{R}$. In [9] it is shown that there are regular processes that cannot be defined just using iteration. Just having prefix iteration, the difference can be found even more simply, consider e.g. $X = abX$. In the model $\mathbf{P}$, the principle RSP boils down to the following conditional axiom:

$$x = ax + y \implies x = a^*y \qquad \text{RSP}^*.$$

5 Parallel Composition

We extend the theory SPA with parallel composition and communication, by adding the following signature elements.

- First of all, we assume that on the set of actions A, we have given a partial binary function γ that is commutative and associative (when defined). For actions $a, b \in A$, if $\gamma(a, b) = c$ is defined, this means that the simultaneous execution of actions a and b (by parallel components) yields the communication c (between the components). The *communication function* γ is a parameter of the theory.
- We have the binary operator $\|$ denoting parallel composition or *merge*. In order to provide a finite axiomatization, we have the auxiliary operators $\lfloor\!\lfloor$ *left merge* and $|$ *communication merge*. The merge axiom MC expresses that $x \| y$ can start with a step from x, a step from y or a communication between x and y.
- The left merge $x \lfloor\!\lfloor y$ behaves just like $x \| y$, with the restriction that the first step is a step from x.
- The communication merge $x \mid y$ behaves just like $x \| y$, with the restriction that the first step is a communication step between x and y.

- The *encapsulation operator* ∂_H from Section 3 will block the execution of actions from the set of actions H. It is used to enforce communication between components.

The axioms of the resulting theory IPA (Interaction Process Algebra) consist of the axioms in Tables 1, 5 and in Table 10. If we delete the last summand of axiom MC, the axioms in Table 5 and axioms CM1-6, we obtain the theory CPA, Concurrent Process Algebra, containing parallel composition but without communication.

$x \parallel y = x \between y + y \between x + x \mid y$	MC		
$\delta \between x = \delta$	LM1	$x \mid y = y \mid x$	CM1
$\epsilon \between \delta = \delta$	LM2	$\delta \mid x = \delta$	CM2
$\epsilon \between \epsilon = \epsilon$	LM3	$\epsilon \mid x = \delta$	CM3
$\epsilon \between ax = \delta$	LM4		
$\epsilon \between (x + y) = \epsilon \between x + \epsilon \between y$	LM5	$ax \mid by = c(x \parallel y)$ if $\gamma(a,b) = c$	CM4
$ax \between y = a(x \parallel y)$	LM6	$ax \mid by = \delta$ if $\gamma(a,b)$ undefined	CM5
$(x + y) \between z = x \between z + y \between z$	LM7	$(x + y) \mid z = x \mid z + y \mid z$	CM6

Table 10: Axioms of IPA ($a, b, c \in A$).

The axiomatization follows the structure of MPA-terms. For the left-merge, we look at the form of the left-hand argument. In case this is ϵ, we have to look at the form of the right-hand argument. The axioms for $\epsilon \between x$ together code the termination behaviour of merge: a parallel composition can only terminate if both arguments can terminate, i.e. both arguments have an ϵ summand. As a result, we have

$$\epsilon \between x = \begin{cases} \epsilon & \text{if } x = x + \epsilon \\ \delta & \text{otherwise.} \end{cases}$$

For the axioms of communication merge, we first assume commutativity. This implies the commutativity of merge, a property that we want to have in any case. Then, we have an axiom in case one of the arguments is a constant or a sum. The remaining case is now where both arguments have the form of an action prefix, and then we have a case distinction, depending on whether the communication between the two actions is defined. Again, due to our modifications of the standard theory, we have a clear separation between the communication *function* and the communication *operator*. We could have also included an axiom for the associativity of the merge, as we want this to hold in every model we consider. The following proposition shows the associativity holds on the initial algebra.

Proposition 8 For all closed IPA-terms x, y, z, we have the following equations (called the axioms of *Standard Concurrency*):

$$x \parallel (y \parallel z) = (x \parallel y) \parallel z$$
$$x \between (y \parallel z) = (x \between y) \between z$$
$$x \mid (y \between z) = (x \mid y) \between z$$
$$x \mid (y \mid z) = (x \mid y) \mid z$$

The proof is as usual. As a consequence of this proposition, we can formulate an expansion theorem.

The deduction rules for IPA add the rules in Tables 11 and 6 to the rules in Table 2. Deleting the rules in the lower half of Table 11 yield the rules for CPA. Due to the absence of the $. \xrightarrow{a} \surd$ predicate, there are fewer deduction rules than for standard PA or ACP.

We have an elimination theorem for both CPA and IPA: each closed term can be reduced to an MPA-term. Using this theorem we can show that axiom CM1 can be derived for all closed terms from the other axioms.

Next, we can show that CPA and IPA are both conservative extensions of SPA, and the axiomatizations are sound and complete for the model of transition systems modulo bisimulation equivalence.

$$\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} \qquad \frac{x\downarrow, y\downarrow}{x \parallel y\downarrow}$$

$$\frac{x \xrightarrow{a} x'}{x \between y \xrightarrow{a} x' \parallel y} \qquad \frac{x\downarrow, y\downarrow}{x \between y\downarrow}$$

$$\frac{x \xrightarrow{a} x',\ y \xrightarrow{b} y',\ \gamma(a,b)=c}{x \parallel y \xrightarrow{c} x' \parallel y'} \qquad \frac{x \xrightarrow{a} x',\ y \xrightarrow{b} y',\ \gamma(a,b)=c}{x \mid y \xrightarrow{c} x' \parallel y'}$$

Table 11: Deduction rules for IPA ($a, b, c \in A$).

We can add prefix iteration to CPA or to IPA, giving a conservative extension of these theories, but then we can no longer obtain a complete axiomatization: we can reduce the expression $\partial_H(a^*x)$, but have problems with an expression of the form $a^*x \between y$. We can formulate an expansion theorem, nevertheless, when we extend action prefix iteration to a prefix iteration operator of a *sum* of actions, the so-called *flat* iteration of [15].

6 Silent Step

We consider the silent step τ. We choose to treat the silent step as an action, so we take $\tau \in A$. This means we have the action prefix τx as a particular prefix operator. This emphasises the fact that τ is an action (whose execution cannot be observed) and as such has nothing to do with (some form of) termination, τ and ϵ can be clearly distinguished. The process $\tau\epsilon$ will terminate without executing a visible action, the process $\tau\delta$ can be called deadlock: without executing a visible action, a state will be reached where the process is stuck.

As semantical treatment of the silent step we choose rooted branching bisimulation (see [16]) rather than Milner's original weak bisimulation (see e.g. [18]), as the former is closer to an action interpretation, and all axioms of IPA that hold for all actions also hold for τ.

In a setting with ϵ, there is only one extra axiom for τ, the branching axiom B shown in Table 12. Taking $y = x$, we obtain $a\tau x = ax$, so every τ-step that is not the first step and is not part of a sum can be removed.

The theories MPA_τ, SPA_τ, CPA_τ, IPA_τ are obtained from the theories MPA, SPA, CPA and IPA, resp. by having a special element $\tau \in A$ and adding axiom B. We can obtain

$a(\tau(x+y)+x) = a(x+y)$ B

Table 12: Axiom of silent step ($a \in A$).

an elimination theorem as before, all closed terms can be reduced to MPA-terms. In the semantics, we cannot capture the special behaviour of the silent step in terms of deduction rules. Rather, we have to divide out a different equivalence relation on the transition systems generated by the rules we have defined previously.

Definition 9 For closed terms s, t, we define $s \Rightarrow t$ if t can be reached from s by doing a number of τ-steps (0 or more). Moreover, we put $s \stackrel{(a)}{\rightarrow} t$ if either $a = \tau$ and $s = t$ or $s \stackrel{a}{\rightarrow} t$.

Then, let R be a binary *symmetric* relation on closed terms. We say R is a *branching bisimulation* if the following holds:

- whenever $R(x, y)$ and $x \stackrel{a}{\rightarrow} x'$ then there are terms y'', y' such that $y \Rightarrow y'' \stackrel{(a)}{\rightarrow} y'$ and $R(x, y'')$ and $R(x', y')$
- whenever $R(x, y)$ and $x \downarrow$, then there is a term y'' such that $y \Rightarrow y'' \downarrow$ and $R(x, y'')$

If R is a branching bisimulation relating terms s, t then we say s, t satisfy the *root condition* (for R) if the following holds:

- whenever $s \stackrel{a}{\rightarrow} x$ then there is a term y such that $t \stackrel{a}{\rightarrow} y$ and $R(x, y)$
- whenever $t \stackrel{a}{\rightarrow} y$ then there is a term x such that $s \stackrel{a}{\rightarrow} x$ and $R(x, y)$
- $s \downarrow$ iff $t \downarrow$

We say two closed terms t, s are *rooted branching bisimulation equivalent* or *rooted branching bimilar*, notation $t \leftrightarrows_{rb} s$ if there is a branching bisimulation R with $R(s, t)$ with satisfies the root condition for s, t.

We can prove that rooted branching bisimulation equivalence is a congruence relation on IPA_τ-terms. We obtain models for the various theories with complete axiomatizations.

Theorem 10 *Let X be one of the theories $MPA_\tau, SPA_\tau, CPA_\tau, IPA_\tau$ and let s, t be closed X-terms.*

Then $X \vdash s = t \iff s \leftrightarrows_{rb} t$.

Proof Follow [16], see also [8]. □

Next, let us consider the other extensions. When we consider renamings, we have to keep τ fixed, i.e. we can only allow renaming functions $f : A \rightarrow A$ that satisfy $f(\tau) = \tau$. There is one renaming operator that is particularly useful and that is the *abstraction operator*, usually denoted τ_I, that renames all actions from $I \subseteq A$ into τ, and is based on the following function f:

$$f(a) = \begin{cases} \tau & \text{if } a \in I \\ a & \text{otherwise.} \end{cases}$$

Considering prefix iteration, we get the following. The τ prefix iteration can be called the *divergence* prefix, and we can formulate the *fair iteration* axiom FI. See Table 13.

$$\tau\tau^*x = \tau x \quad \text{FI}$$

Table 13: Axioms for divergence ($a \in A$).

Note that the fair iteration axiom is equivalent to $\tau^*x = \tau x + x$. It is valid on the model of transition systems modulo rooted branching bisimulation equivalence. However, we have to be careful with τ and recursion, since τ cannot serve as a guard. Put another way, variables can only be considered guarded if they are in the scope of an action prefix for an action different from τ. To give an example, if p is any process then we can derive $\tau p = \tau\tau p$, so any process of the form τp is a solution of the equation $X = \tau X$ and the principle RSP* is not valid for $a = \tau$.

7 Discrete Time Process Algebra with Relative Timing

So far, we have considered untimed process algebra. We see the theory can be developed in the usual way, although due to our reformulation of the basic theory, several improvements could be made. The main reason for the reformulation becomes evident, however, when we consider timing. It is not necessary to look at the whole framework of discrete and dense timed, absolute and relative timed, time-stamped and two phase process algebras (see [5]). Instead, we can make the point by considering one member of this family, viz. process algebra with discrete time in relative timing in two phase notation (see [4]). Also, it is sufficient to consider the theory without parallel composition.

We have the following syntax in addition to signature elements of SPA:

- *current time slice action prefix* $\underline{a}$, where $a \in A$. The process $\underline{a}x$ will execute action a in the current time slice and evolve into x.
- *current slice time stop* $\underline{\delta}$. Time cannot progress beyond the current time slice, and no termination can take place. Notice that we do not include the constant $\dot{\delta}$, for which the current moment of time is already inconsistent. We can omit this second constant, since we only consider relative timing. In the absence of $\dot{\delta}$, $\underline{\delta}$ is the neutral element of alternative composition.
- *current slice termination* $\underline{\epsilon}$. Time cannot progress beyond the current time slice, and termination takes place. The constant $\underline{\epsilon}$ is the neutral element of sequential composition.
- *time prefix* σ. The process σx will pass to the next time slice and then execute x. We elect to take $\sigma \notin A$ (this decision is rather arbitrary but emphasises the difference between passage of time and execution of an atomic action).

We first consider a version of the theory without the untimed prefix operators $a.$, and the untimed constants δ, ϵ, indicated by a superscript $^-$ after the algebra name. The axiomatisation of MPA^-_{drt} adds axiom DRTF and A6DR in Table 14 to axioms A1,2,3 of MPA in

Table 1, the theory SPA^{-}_{drt} adds all axioms in Table 14 to axioms A1-5 and A10 of SPA in Table 1.

The *Time Factorization* axiom DRTF expresses that the choice in alternative composition $+$ is resolved by the execution of an action, not by the mere passage of time. For sequential composition $\cdot$, relative timing means that time is measured from the execution of the previous action. This is expressed by axiom A10S.

$\underline{\underline{a}}x \cdot y = \underline{\underline{a}}(x \cdot y)$	A10DR	$x + \underline{\underline{\delta}} = x$	A6DR
$\sigma x \cdot y = \sigma(x \cdot y)$	A10S	$\underline{\underline{\delta}} \cdot x = \underline{\underline{\delta}}$	A7DR
$\sigma x + \sigma y = \sigma(x + y)$	DRTF	$\underline{\underline{\epsilon}} \cdot x = x$	A8DR
		$x \cdot \underline{\underline{\epsilon}} = x$	A9DR

Table 14: Axioms of SPA in relative discrete time ($a \in A$).

The definition of an operational semantics by means of SOS deduction rules is straightforward. To the relations of Table 2, we add a binary relation $. \overset{1}{\mapsto} .$ on closed terms. Intuitively, $x \overset{1}{\mapsto} x'$ means that x evolves into x' by passing to the next time slice. We add the rules in Table 15 to the rules of Table 2. Note that $x \overset{1}{\not\mapsto}$ means that x cannot execute a $\overset{1}{\mapsto}$ transition, i.e. x cannot pass to the next time slice. Thus, we have here an SOS definition with negative premises. It is well-defined, however, as is shown in [21]. Using the technique of *saturation*, it is possible to avoid the negative premises, see [6].

$$\underline{\underline{a}}x \overset{a}{\rightarrow} x \qquad \underline{\underline{\epsilon}}\downarrow \qquad \sigma x \overset{1}{\mapsto} x$$

$$\frac{x \overset{1}{\mapsto} x',\ y \overset{1}{\mapsto} y'}{x + y \overset{1}{\mapsto} x' + y'} \qquad \frac{x \overset{1}{\mapsto} x',\ y \overset{1}{\not\mapsto}}{x + y \overset{1}{\mapsto} x'} \qquad \frac{y \overset{1}{\mapsto} y',\ x \overset{1}{\not\mapsto}}{x + y \overset{1}{\mapsto} y'}$$

$$\frac{x \overset{1}{\mapsto} x',\ x \not\downarrow}{x \cdot y \overset{1}{\mapsto} x' \cdot y} \qquad \frac{x \overset{1}{\mapsto} x',\ y \overset{1}{\not\mapsto}}{x \cdot y \overset{1}{\mapsto} x' \cdot y} \qquad \frac{x \overset{1}{\not\mapsto},\ x \downarrow,\ y \overset{1}{\mapsto} y'}{x \cdot y \overset{1}{\mapsto} y'} \qquad \frac{x \overset{1}{\mapsto} x',\ x \downarrow,\ y \overset{1}{\mapsto} y'}{x \cdot y \overset{1}{\mapsto} x' \cdot y + y'}$$

Table 15: Deduction rules for SPA with relative discrete time ($a \in A$).

Next, we want to extend this theory in order to embed the untimed action prefix operators $a.$ and the untimed termination constants ϵ, δ also, thereby having a full embedding of SPA into SPA_{drt}. We interpret an action prefix $a.$ as a commitment to execute a at some unspecified time, i.e. an arbitrary number of time steps can occur before the execution of a. Likewise, ϵ stands for successful termination at some unspecified time, i.e. after an arbitrary number of time steps, and δ cannot execute actions or terminate, so can only let time pass.

The axiomatization is easiest if first, we introduce the time iteration prefix σ^*. The time iteration prefix will prefix the process that follows with an arbitrary number of time steps. Given time iteration, axiomatization of $a., \epsilon, \delta$ is easy, see Table 16. The operational rules are straightforward, and presented in Table 17.

$\epsilon = \sigma^*\underline{\underline{\epsilon}}$	$\sigma(\sigma^* x) + x = \sigma^* x$
$\delta = \sigma^*\underline{\underline{\delta}}$	$\sigma^*(\sigma^* x) = \sigma^* x$
$ax = \sigma^*(\underline{\underline{a}}x)$	$\sigma^* x \cdot y = \sigma^*(x \cdot y)$
	$\sigma^* x + \sigma^* y = \sigma^*(x + y)$
	$\sigma\sigma^* x = \sigma^*\sigma x$

Table 16: Embedding untimed actions, time prefix iteration ($a \in A$).

$$ax \overset{1}{\mapsto} ax \qquad \delta \overset{1}{\mapsto} \delta \qquad \epsilon \overset{1}{\mapsto} \epsilon$$

$$ax \xrightarrow{a} x \qquad \epsilon\downarrow$$

$$\frac{x \xrightarrow{a} x'}{\sigma^* x \xrightarrow{a} x'} \qquad \frac{x\downarrow}{\sigma^* x\downarrow} \qquad \frac{x \overset{1}{\not\mapsto}}{\sigma^* x \overset{1}{\mapsto} \sigma^* x} \qquad \frac{x \overset{1}{\mapsto} x'}{\sigma^* x \overset{1}{\mapsto} x' + \sigma^* x}$$

Table 17: Deduction rules for untimed embedding and time prefix iteration ($a \in A$).

We can now appreciate the advantage of having actions as a prefixing operator instead of a constant: we have the desired embedding $ax = \sigma^*(\underline{\underline{a}}x)$, and we can consider different forms of termination: $a\underline{\underline{\epsilon}}$ will terminate in the same time slice as the execution of a, whereas $a\epsilon$ will terminate some unspecified time after the execution of a. We also have the forms $\underline{\underline{a\epsilon}}$, both action execution and termination in the current time slice, and $\underline{\underline{a}}\epsilon$, action execution in the current time slice, termination at some later time.

In the old setting, the interpretation of a term $ab\underline{\underline{x}}$ must be that b is executed in the same time slice as a, so the interpretation of the (untimed) constant process a in the timed theory must include immediate termination (or at least, current time slice termination). But then, the equation $a \cdot \epsilon = a$ is not valid anymore, spoiling the equations of the untimed theory (see [6]). It is this difficulty, that lead us to develop the current paper. Consequently, several other advantages of action prefix over action constants were found.

8 Conclusion

We have presented a redesign of ACP-style process algebra, where action execution and termination are separated. This allows an improved embedding of untimed into timed process algebra.

The separation of action execution and termination also entails better separation of atomic actions as a parameter of the theory and as signature elements. We can define a minimal process algebra without sequential composition, and this makes formulation of concepts such as structural induction, linearity, elimination and guardedness easier. The difference between the silent step τ and the empty step ϵ becomes clearer.

In the operational semantics, we have no need for separate terminating action executions.

We have a natural restriction of iteration to action prefix iteration, allowing complete axiomatizations in more cases. Moreover, prefix iteration is sufficient to formulate time iteration and embedding of untimed into timed theories, and sufficient to formulate divergent behaviour and fair iteration.

References

[1] L. Aceto and J.F. Groote. A complete equational axiomatization for MPA with string iteration. *Theoretical Computer Science*, 211(1/2):339–374, 1999.

[2] J.C.M. Baeten and J.A. Bergstra. On sequential composition, action prefixes and process prefix. *Formal Aspects of Computing*, 6(3):250–268, 1994.

[3] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra with abstraction. In H. Reichel, editor, *Proceedings FCT'95*, number 965 in Lecture Notes in Computer Science, pages 1–15, Dresden, 1995. Springer Verlag.

[4] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. *Formal Aspects of Computing*, 8(2):188–208, 1996.

[5] J.C.M. Baeten and C.A. Middelburg. Process algebra with timing: Real time and discrete time. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*. Elsevier Science, Amsterdam, 2000. To appear.

[6] J.C.M. Baeten and M.A. Reniers. Termination in timed process algebra. Technical Report CSR 00/??, Eindhoven University of Technology, Computing Science Department, 2000.

[7] J.C.M. Baeten and C. Verhoef. Concrete process algebra. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 149–269. Oxford University Press, 1995.

[8] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[9] J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *The Computer Journal*, 37(4):243–258, 1994.

[10] J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proceedings 11th ICALP*, number 172 in LNCS, pages 82–95. Springer Verlag, 1984.

[11] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.

[12] J.A. Bergstra and J.V. Tucker. Top down design and the algebra of communicating processes. *Science of Computer Programming*, 5:171–199, 1984.

[13] D.J.B. Bosscher. *Grammars Modulo Bisimulation*. PhD thesis, University of Amsterdam, 1997.

[14] W.J. Fokkink. A complete equational axiomatisation for prefix iteration. *Information Processing Letters*, 52(6):333–337, 1994.

[15] R.J. van Glabbeek. Axiomatizating flat iteration. In A. Mazurkiewicz and J. Winkowski, editors, *Proceedings CONCUR'97*, number 1243 in Lecture Notes in Computer Science, pages 228–242. Springer Verlag, 1997.

[16] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[17] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[18] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[19] G.D. Plotkin. An operational semantics for CSP. In D. Bjørner, editor, *Proceedings Conference on Formal Description of Programming Concepts II*, pages 199–225. North-Holland, Amsterdam, 1983.

[20] P. Sewell. Nonaxiomatisability of equivalences over finite state processes. *Annals of Pure and Applied Logic*, 90:163–191, 1997.

[21] J.J. Vereijken. *Discrete-Time Process Algebra.* PhD thesis, Eindhoven University of Technology, 1997.

Computing Science Reports

Department of Mathematics and Computing Science
Eindhoven University of Technology

In this series appeared:

96/01	M. Voorhoeve and T. Basten	Process Algebra with Autonomous Actions, p. 12.
96/02	P. de Bra and A. Aerts	Multi-User Publishing in the Web: DreSS, A Document Repository Service Station, p. 12
96/03	W.M.P. van der Aalst	Parallel Computation of Reachable Dead States in a Free-choice Petri Net, p. 26.
96/05	T. Basten and W.M.P. v.d. Aalst	A Process-Algebraic Approach to Life-Cycle Inheritance Inheritance = Encapsulation + Abstraction, p. 15.
96/06	W.M.P. van der Aalst and T. Basten	Life-Cycle Inheritance A Petri-Net-Based Approach, p. 18.
96/07	M. Voorhoeve	Structural Petri Net Equivalence, p. 16.
96/08	A.T.M. Aerts, P.M.E. De Bra, J.T. de Munk	OODB Support for WWW Applications: Disclosing the internal structure of Hyperdocuments, p. 14.
96/09	F. Dignum, H. Weigand, E. Verharen	A Formal Specification of Deadlines using Dynamic Deontic Logic, p. 18.
96/10	R. Bloo, H. Geuvers	Explicit Substitution: on the Edge of Strong Normalisation, p. 13.
96/11	T. Laan	AUTOMATH and Pure Type Systems, p. 30.
96/12	F. Kamareddine and T. Laan	A Correspondence between Nuprl and the Ramified Theory of Types, p. 12.
96/13	T. Borghuis	Priorean Tense Logics in Modal Pure Type Systems, p. 61
96/14	S.H.J. Bos and M.A. Reniers	The I^2C-bus in Discrete-Time Process Algebra, p. 25.
96/15	M.A. Reniers and J.J. Vereijken	Completeness in Discrete-Time Process Algebra, p. 139.
96/17	E. Boiten and P. Hoogendijk	Nested collections and polytypism, p. 11.
96/18	P.D.V. van der Stok	Real-Time Distributed Concurrency Control Algorithms with mixed time constraints, p. 71.
96/19	M.A. Reniers	Static Semantics of Message Sequence Charts, p. 71
96/20	L. Feijs	Algebraic Specification and Simulation of Lazy Functional Programs in a concurrent Environment, p. 27.
96/21	L. Bijlsma and R. Nederpelt	Predicate calculus: concepts and misconceptions, p. 26.
96/22	M.C.A. van de Graaf and G.J. Houben	Designing Effective Workflow Management Processes, p. 22.
96/23	W.M.P. van der Aalst	Structural Characterizations of sound workflow nets, p. 22.
96/24	M. Voorhoeve and W. van der Aalst	Conservative Adaption of Workflow, p.22
96/25	M. Vaccari and R.C. Backhouse	Deriving a systolic regular language recognizer, p. 28
97/02	J. Hooman and O. v. Roosmalen	A Programming-Language Extension for Distributed Real-Time Systems, p. 50.
97/03	J. Blanco and A. v. Deursen	Basic Conditional Process Algebra, p. 20.
97/04	J.C.M. Baeten and J.A. Bergstra	Discrete Time Process Algebra: Absolute Time, Relative Time and Parametric Time, p. 26.
97/05	J.C.M. Baeten and J.J. Vereijken	Discrete-Time Process Algebra with Empty Process, p. 51.
97/06	M. Franssen	Tools for the Construction of Correct Programs: an Overview, p. 33.
97/07	J.C.M. Baeten and J.A. Bergstra	Bounded Stacks, Bags and Queues, p. 15.
97/08	P. Hoogendijk and R.C. Backhouse	When do datatypes commute? p. 35.

97/09	Proceedings of the Second International Workshop on Communication Modeling, Veldhoven, The Netherlands, 9-10 June, 1997.	Communication Modeling- The Language/Action Perspective, p. 147.
97/10	P.C.N. v. Gorp, E.J. Luit, D.K. Hammer E.H.L. Aarts	Distributed real-time systems: a survey of applications and a general design model, p. 31.
97/11	A. Engels, S. Mauw and M.A. Reniers	A Hierarchy of Communication Models for Message Sequence Charts, p. 30.
97/12	D. Hauschildt, E. Verbeek and W. van der Aalst	WOFLAN: A Petri-net-based Workflow Analyzer, p. 30.
97/13	W.M.P. van der Aalst	Exploring the Process Dimension of Workflow Management, p. 56.
97/14	J.F. Groote, F. Monin and J. Springintveld	A computer checked algebraic verification of a distributed summation algorithm, p. 28
97/15	M. Franssen	λP-: A Pure Type System for First Order Loginc with Automated Theorem Proving, p.35.
97/16	W.M.P. van der Aalst	On the verification of Inter-organizational workflows, p. 23
97/17	M. Vaccari and R.C. Backhouse	Calculating a Round-Robin Scheduler, p. 23.
97/18	Werkgemeenschap Informatiewetenschap redactie: P.M.E. De Bra	Informatiewetenschap 1997 Wetenschappelijke bijdragen aan de Vijfde Interdisciplinaire Conferentie Informatiewetenschap, p. 60.
98/01	W. Van der Aalst	Formalization and Verification of Event-driven Process Chains, p. 26.
98/02	M. Voorhoeve	State / Event Net Equivalence, p. 25
98/03	J.C.M. Baeten and J.A. Bergstra	Deadlock Behaviour in Split and ST Bisimulation Semantics, p. 15.
98/04	R.C. Backhouse	Pair Algebras and Galois Connections, p. 14
98/05	D. Dams	Flat Fragments of CTL and CTL*: Separating the Expressive and Distinguishing Powers. P. 22.
98/06	G. v.d. Bergen, A. Kaldewaij V.J. Dielissen	Maintenance of the Union of Intervals on a Line Revisited, p. 10.
98/07	Proceedings of the workhop on Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98) June 22, 1998 Lisbon, Portugal	edited by W. v.d. Aalst, p. 209
98/08	Informal proceedings of the Workshop on User Interfaces for Theorem Provers. Eindhoven University of Technology ,13-15 July 1998	edited by R.C. Backhouse, p. 180
98/09	K.M. van Hee and H.A. Reijers	An analytical method for assessing business processes, p. 29.
98/10	T. Basten and J. Hooman	Process Algebra in PVS
98/11	J. Zwanenburg	The Proof-assistemt Yarrow, p. 15
98/12	Ninth ACM Conference on Hypertext and Hypermedia Hypertext '98 Pittsburgh, USA, June 20-24, 1998 Proceedings of the second workshop on Adaptive Hypertext and Hypermedia.	Edited by P. Brusilovsky and P. De Bra, p. 95.
98/13	J.F. Groote, F. Monin and J. v.d. Pol	Checking verifications of protocols and distributed systems by computer. Extended version of a tutorial at CONCUR'98, p. 27.
98/14	T. Verhoeff (artikel volgt)	
99/01	V. Bos and J.J.T. Kleijn	Structured Operational Semantics of χ , p. 27
99/02	H.M.W. Verbeek, T. Basten and W.M.P. van der Aalst	Diagnosing Workflow Processes using Woflan, p. 44
99/03	R.C. Backhouse and P. Hoogendijk	Final Dialgebras: From Categories to Allegories, p. 26
99/04	S. Andova	Process Algebra with Interleaving Probabilistic Parallel Composition, p. 81

99/05	M. Franssen, R.C. Veltkamp and W. Wesselink	Efficient Evaluation of Triangular B-splines, p. 13
99/06	T. Basten and W. v.d. Aalst	Inheritance of Workflows: An Approach to tackling problems related to change, p. 66
99/07	P. Brusilovsky and P. De Bra	Second Workshop on Adaptive Systems and User Modeling on the World Wide Web, p. 119.
99/08	D. Bosnacki, S. Mauw, and T. Willemse	Proceedings of the first international syposium on Visual Formal Methods - VFM'99
99/09	J. v.d. Pol, J. Hooman and E. de Jong	Requirements Specification and Analysis of Command and Control Systems
99/10	T.A.C. Willemse	The Analysis of a Conveyor Belt System, a case study in Hybrid Systems and timed μ CRL, p. 44.
99/11	J.C.M. Baeten and C.A. Middelburg	Process Algebra with Timing: Real Time and Discrete Time, p. 50.
99/12	S. Andova	Process Algebra with Probabilistic Choice, p. 38.
99/13	K.M. van Hee, R.A. van der Toorn, J. van der Woude and P.A.C. Verkoulen	A Framework for Component Based Software Architectures, p. 19
99/14	A. Engels and S. Mauw	Why men (and octopuses) cannot juggle a four ball cascade, p. 10
99/15	J.F. Groote, W.H. Hesselink, S. Mauw, R. Vermeulen	An algorithm for the asynchronous *Write-All* problem based on process collision*, p. 11.
99/16	G.J. Houben, P. Lemmens	A Software Architecture for Generating Hypermedia Applications for Ad-Hoc Database Output, p. 13.
99/17	T. Basten, W.M.P. v.d. Aalst	Inheritance of Behavior, p.83
99/18	J.C.M. Baeten and T. Basten	Partial-Order Process Algebra (and its Relation to Petri Nets), p. 79
99/19	J.C.M. Baeten and C.A. Middelburg	Real Time Process Algebra with Time-dependent Conditions, p.33.
99/20	Proceedings Conferentie Informatiewetenschap 1999 Centrum voor Wiskunde en Informatica 12 november 1999, p.98	edited by P. de Bra and L. Hardman
00/01	J.C.M. Baeten and J.A. Bergstra	Mode Transfer in process Algebra, p. 14