EINDHOVEN UNIVERSITY OF TECHNOLOGY

A process algebra of concurrent constraint programming

Citation for published version (APA):
Boer, de, F. S., & Palamidessi, C. (1993). *A process algebra of concurrent constraint programming*. (Computing science notes; Vol. 9320). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/1993

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 16. Nov. 2023

Eindhoven University of Technology

Department of Mathematics and Computing Science

A Process Algebra of Concurrent Constraint Programming

by

F.S. de Boer

93/20

Computing Science Note 93/20
Eindhoven, June 1993

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

All rights reserved
editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

A Process Algebra of Concurrent Constraint Programming

Frank S. de Boer
Department of Computing Science, Technical University Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
wsinfdb@tuewsd.win.tue.nl

Catuscia Palamidessi
Department of Computer Science, University of Pisa,
Corso Italia 40, 56100 Pisa, Italy
katuscia@apollo.di.unipi.it

Abstract

We develop an algebraic theory for the observational equivalence of concurrent constraint programs which identifies processes which have the same final results for all possible executions.

1 Introduction

In the last years there have been given several proposals to extend logic programming with constructs for concurrency, aiming at the development of a concurrent language which would maintain the typical advantages of logic programming: declarative reading, computations as proofs, amenability to meta-programming etc. Examples of concurrent logic languages include PARLOG [6], Concurrent Prolog [12, 13], Guarded Horn Clauses [16, 17] and their so-called *flat* versions. Concurrent constraint programming ([10, 14, 15]) represents one of the most successful proposals in this area.

Constraint programming is based on the notion of computing with systems of partial information. The *store* is seen as a constraint on the values that variables can assume, rather than a correspondence between variables and values. All processes of the system share a common store, which, at any stage of the computation, is given by the constraint established until that moment. The execution of a tell action modifies the current store by adding a constraint. An ask action is a test on the store: it can be executed only if the current store is *strong enough* to entail a specified constraint. If this is not the case, then the process suspends (waiting for the store to accumulate more information by the contributions of the other processes). The execution of an ask itself leaves the store unchanged. Hence both the tell and ask actions are monotonic, in the sense that after their execution the store contains the same or more information. Therefore the store evolves monotonically during the computation, i.e. the set of possible values for the

variables shrinks.

This paper addresses the problem of an algebraic axiomatization for concurrent constraint programming. The algebraic approach is one of the most diffused methods in concurrency theory both for specification (i.e. definition of new operators) and for program verification (i.e. check that a certain implementation satisfies a given specification). During the last decade there have been a number of proposals for process algebras: beside the Calculus of Communicating Systems of Milner ([9]), several related formalisms have been proposed, such as the Theory of Communicating Processes of Hoare ([1]) and the Algebra of Communicating Systems of Bergstra and Klop ([2, 3, 5]).

For a given language there are, in general, various observability criteria which are of interest. Since in concurrent constraint programming processes communicate via a common store the relevant aspects of the behaviour of a process, from the point of view of the environment, are described in terms of its interaction with the common store. In this paper we consider the most abstract description: only the final results are observable. This choice is motivated by the fact that, due to the monotonic evolution of the store, the intermediate states of the computation are just approximations of the final result.

However, the equivalence induced by this notion of observables introduces too many identifications to be characterized algebraically. For an algebraic theory only those processes can be identified which not only have the same observables, but which additionally show no observable difference when immerged in any kind of context. An equivalence which satisfies this property is called a *congruence*. The coarsest of such congruences is particularly of interest since it exactly identifies those processes which cannot be distinguished by any context; it corresponds to a fully abstract semantics. In this paper we will develop a complete axiomatization of the coarsest congruence contained in the equivalence induced by observing final results only.

To prove correctness and completeness of the axiomatization, it will be convenient to define a fully abstract semantics. A compositional model is more suitable for reasoning about the axioms because it characterizes classes of processes which are observable equivalent in every context in terms of canonical representatives.

Due to space limitations in this version we have omitted the proofs which can be found in the full paper.

1.1 Plan of the paper

In the next section we define the notion of constraint system underlying the language. In particular, we discuss distributive and complemented constraint systems. In Section 3 we define the language, the operational model, and the notion of observables. In Section 4 we present the axiomatization, and in Section 5 we discuss its correctness and completeness. In section 5 we also develop a fully abstract semantics which will be useful to prove those

results. In the last section we point out some directions for future research.

2 Constraint systems

The notion of constraint system we consider here is a simplification[1] of the one developed in [14].

Definition 2.1 A constraint system $\mathcal{C}$ is a complete (algebraic) lattice $\langle C, \leq, \wedge, \mathit{true}, \mathit{false}\rangle$ where $\wedge$ is the lub operation, and *true*, *false* are the least and the greatest elements of C, respectively.

Following the standard terminology and notation, instead of $\leq$ we will refer to its inverse relation, denoted by $\vdash$ and called *entailment*. Formally

$$\forall c, d \in C.\ c \vdash d \Leftrightarrow d \leq c.$$

In order to treat the hiding operator of the language it will be helpful to introduce a general notion of existential quantification. In this framework it is convenient to formalize this notion by means of the theory of cylindric algebras ([8]). This leads to the concept of *cylindric constraint system.*

Definition 2.2 Let *Var* be a (denumerable) set of variables $x, y, z, \ldots$. Assume that for each $x \in \mathit{Var}$ a function $\exists_x : C \rightarrow C$ is defined such that for any $c, d \in C$:

(i) $c \vdash \exists_x(c)$,

(ii) if $c \vdash d$ then $\exists_x(c) \vdash \exists_x(d)$,

(iii) $\exists_x(c \wedge \exists_x(d)) \sim \exists_x(c) \wedge \exists_x(d)$,

(iv) $\exists_x(\exists_y(c)) \sim \exists_y(\exists_x(c))$.

Then $\langle C, \vdash, \wedge, \mathit{true}, \mathit{false}, \mathit{Var}\rangle$ is a *cylindric constraint system.*

In the following $\exists_x(c)$ will be denoted by $\exists_x c$ with the convention that, in case of ambiguity, the scope of $\exists_x$ is limited to the first constraint subexpression. (So, for instance $\exists_x c \wedge d$ stands for $\exists_x(c) \wedge d$.)

We introduce now two notions taken from lattice theory: the complement and the distributivity. The complement of an element c, denoted by c^-, represents, in a sense, the negation of c. Distributivity is the usual property about combinations of lubs and glbs (in the sequel the glb of a lattice will be

[1]The approach of [14] follows Scott's treatment of information system ([11]): the starting point is a set of simple constraints on which a compact entailment relation is defined. Then a constraint system is constructed by considering sets of simple constraints and by extending the entailment relation on it. This construction is made in such a way that the resulting structure is a complete *algebraic* lattice, which ensures the effectiveness of the extended entailment relation. In this paper we abstract from this construction, and we just consider the resulting structure.

denoted by $\vee$). Constraint systems satisfying distributivity and existence of the complement are very rich structures (actually they are boolean algebras), and for this reason they are particularly suitable to reason about equalities. Furthermore, they have a very interesting feature, which will be useful for developing our axiomatization: they are able to *represent the entailment relation* as a constraint of the system itself:

Proposition 2.3 *Let $\langle C, \vdash, \wedge, true, false \rangle$ be a distributive and complemented constraint system. Then*

$$\forall c, d, e \in C. \, (e \wedge c \vdash d) \Leftrightarrow (e \vdash c^- \vee d).$$

We will denote $c^- \vee d$ by $c \rightarrow d$ and $(\exists_x c^-)^-$ by $\forall_x c$.

The only-if part of previous proposition is a sort of 'deduction theorem' for constraint systems.

In general, the existence of the complement and distributivity is a rather strong assumption, and it would be very restrictive to require it to be satisfied by the constraint system on which the language operates. Actually we do not need to do so. For our purpose it is sufficient to embed the constraint system of the language into a complemented and distributive one. We use this larger system only to represent terms in intermediate steps possibly needed to derive certain equalities among processes. (This is in analogy for instance with the idea of immerging the real numbers into the field of the complex numbers, in order to solve equations between real numbers.) Given a constraint system C we will indicate by $dc(C)$ the distributive and complemented closure of C, namely the smallest distributive and complemented constraint system which contains C as subsystem.

Example 2.4 Consider a Herbrand domain consisting only of the constants a, b and c and let C be the constraint system whose elements are the equalities over this domain involving a variable x, and the entailment relation is the 'standard one', represented in Figure 1(a). This constraint system is neither distributive, nor the unicity of the complement is satisfied.

Consider now the constraint system C' which contains also the disequalities involving x, with the 'standard' entailment relation represented in Figure 1(b). We have that C' is distributive, complemented and $C' = dc(C)$.

3 The language

In this section we present the language of concurrent constraint programming, its computational model and the intended observation criterium. The definitions we give are equivalent to the ones in [15].

We assume given a cylindric constraint system $\langle C, \vdash, \wedge, true, false, Var \rangle$. We use $A, B, \ldots$ to range over the set of processes, $p, q, r, \ldots$ to range over

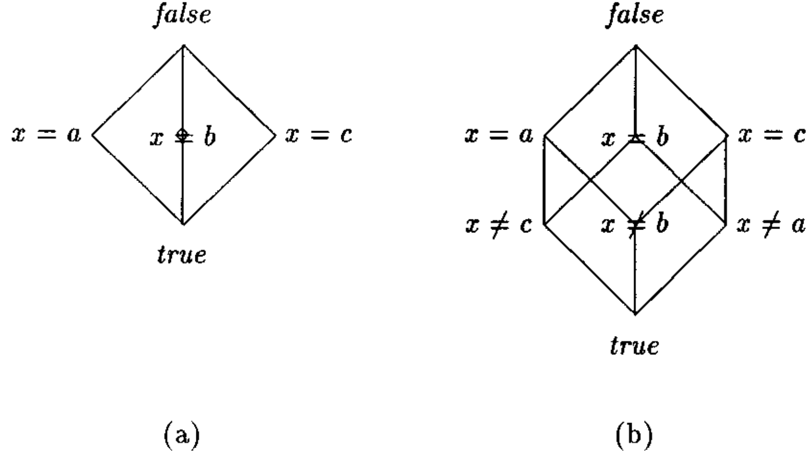

Figure 1: Herbrand constraint systems for x, a, b, c

process names, $x, y, z, \ldots$ to range over *Var* and α over the set of ask and tell actions. In addition, the notation $\vec{\chi}$ indicates a list of the form $(\chi_1, \ldots, \chi_n)$.

The processes are described by the following grammar

$$A ::= \delta \mid \alpha \cdot A \mid A + A \mid A \parallel A \mid \exists x.A \mid p(\vec{x})$$

The symbol δ denotes inaction. The process $\mathbf{ask}(c) \cdot A$ waits until the store entails c and then it behaves like A. The process $\mathbf{tell}(c) \cdot A$ adds c to the store and then it behaves like A. Sometimes we omit $\cdot$ and write, for example, $\alpha(A + B)$ instead of $\alpha \cdot (A + B)$. The operators $\parallel$ and $+$ are the parallel composition (or *merge*) and the nondeterministic choice (or *plus*), respectively. $\exists x$ is the hiding operator: $\exists x.A$ is like the process A, with the variable x seen as *local*. We will see that there is a strong relation with the existential quantifier over the constraint system, for this reason we have used the same symbol. Finally, $p(\vec{x})$ is a procedure call, p is the name of the procedure, and $\vec{x}$ is the list of the actual parameters. The meaning of a process is given with respect to a set W of declarations of the form $p(\vec{y})$:- A. We denote by $Vrt(W)$ the set of the variants of the declarations in W, obtained by renaming their variables. In the sequel we assume W to be fixed, so we omit reference to it.

Syntactical identity between processes we denote by $\equiv$. We assume the following binding order between the operators (corresponding to decreasing priority): $\cdot$, $+$, $\parallel$, $\exists x$.

3.1 The operational model and the observables $\mathcal{O}$

The operational model is described in terms of a transition system $T = (Conf, \longrightarrow)$. The configurations *Conf* consist of a process and a constraint

representing the store. The rules of T are described in Table 1. (We assume the commutativity of the parallel and the choice operator.)

Table 1: The Transition System T

R1 $\quad \langle \mathbf{ask}(d) \cdot A, c\rangle \longrightarrow \langle A, c\rangle \qquad$ **if** $c \vdash d$

R2 $\quad \langle \mathbf{tell}(d) \cdot A, c\rangle \longrightarrow \langle A, c \wedge d\rangle$

R3 $\quad \langle p(\vec{x}), c\rangle \longrightarrow \langle A, c\rangle \qquad$ **if** $p(\vec{x}) \text{ :- } A \in \mathit{Vrt}(W)$

R4
$$\frac{\langle A, \exists_x c\rangle \longrightarrow \langle B, d\rangle}{\langle \exists x.A, c\rangle \longrightarrow \langle \exists_x^d.B, c \wedge \exists_x d\rangle}$$

R5
$$\frac{\langle A, d \wedge \exists_x c\rangle \longrightarrow \langle B, e\rangle}{\langle \exists_x^d.A, c\rangle \longrightarrow \langle \exists_x^e.B, c \wedge \exists_x e\rangle}$$

R6
$$\frac{\langle A, c\rangle \longrightarrow \langle A', d\rangle}{\begin{array}{l}\langle A \parallel B, c\rangle \longrightarrow \langle A' \parallel B, d\rangle \\ \langle A + B, c\rangle \longrightarrow \langle A', d\rangle\end{array}}$$

The way in which the store is queried and updated is described by the rules **R1** and **R2**. Note that the execution of a tell action is not constrained by consistency requirements. As a consequence a tell action can always proceed; it is an autonomous action. Also with respect to an ask action we do not require the current store to be consistent with the asked constraint, we require only that it is implied by the current store. Rule **R3** describes the replacement of a procedure call by the body of the procedure definition (in W). The hiding of variables is described by the rules **R4** and **R5**. To keep track of the local store which contains information about the local variable, we introduced an auxiliary operator $\exists_x^c$, where c represents the local store. A local computation step then proceeds from a store which consists of the local store and the global information about all the variables but the local one. The resulting store of a local computation step represents the new local store, and the new global store is obtained by adding to the old one the new information about all the variables but the local one. Finally, **R6** is the usual rule for the parallel and the choice operator, where the behaviour of a compound process is described in terms of the behaviour of the components. Notice that parallelism is described as interleaving. Furthermore, the choice operator models global non-determinism in the sense that the choices of a process which are guarded by an ask action, depend on the current store which is subject to modifications by the external environment.

The result of a terminating computation consists of the final store. This is formally represented by the notion of *observables.*

Definition 3.1 The observables are given by the function

$$\mathcal{O}[\![A]\!] = \{c \mid \langle A, true\rangle \longrightarrow^* \langle B, c\rangle \not\longrightarrow\}$$

where $\longrightarrow^*$ denotes the transitive closure of $\longrightarrow$, and $\not\longrightarrow$ indicates that there is no transition possible.

We want to identify those processes that have the same obervables in every context:

Definition 3.2 By $\doteq$ we denote the congruence $A \doteq B$ iff for all contexts $C[\,]$, $\mathcal{O}[\![C[A]]\!] = \mathcal{O}[\![C[B]]\!]$. Here a context $C[\,]$ is a process expression with occurrences of a process variable, and $C[A]$ denotes the process obtained by substituting A for this variable in $C[\,]$.

Note that the relation which identifies processes that have the same observables is not a congruence.

4 Process Algebra

In this section we investigate an axiomatization of the congruence $\doteq$. For technical convenience we restrict ourselves to finite processes, for a treatment of recursion we refer to [5]. The kernel of the algebra consists of the axiom system **aprPA** (the system in [4] restricted to action prefixing), plus the failure axioms and the axioms for τ-abstraction ([3]) .

The system **aprPA** (Table 2) axiomatizes the plus-operator (commutativity, associativity, idempotency), δ, and the merge in terms of interleaving. For the axiomatization of the merge an auxiliary operator, the left-merge ($\lfloor\!\lfloor$), is introduced. The system **aprPA** axiomatizes a notion of equivalence which is known as bisimulation [5]. The system **aprPA** plus the failure axioms (Table 3) axiomatizes the congruence induced by the equivalence which identifies processes which have the same maximal traces. Finally, the τ-abstraction rules (Table 4) allow one to abstract from 'silent steps'. In the context of concurrent constraint programming a silent-step corresponds to a **tell**(*true*) or **ask**(*true*) action.

On top of this we have first the axioms for quantification (Table 5). Quantification is axiomatized in terms of the auxiliary operator $\exists_x^c$, which acts like a kind of state-operator [5, 4]. The local store which includes information about the local variable x is represented by c. This auxiliary operator distributes over the plus, and when it passes a tell action or an ask action it quantifies the local variable (in case of an ask it also changes the constraint), and updates the local store, which is then passed on. The transformation of the constraint in the ask can be justified as follows: d is

Table 2: **aprPA**

$\delta \cdot A$	$=$	δ			
$A + A$	$=$	A	$A \parallel B$	$=$	$A \mathbin{\lfloor\!\lfloor} B + B \mathbin{\lfloor\!\lfloor} A$
$A + B$	$=$	$B + A$	$\delta \mathbin{\lfloor\!\lfloor} A$	$=$	δ
$A + (B + C)$	$=$	$(A + B) + C$	$(\alpha \cdot A) \mathbin{\lfloor\!\lfloor} B$	$=$	$\alpha(A \parallel B)$
$A + \delta$	$=$	A	$(A + B) \mathbin{\lfloor\!\lfloor} C$	$=$	$A \mathbin{\lfloor\!\lfloor} C + B \mathbin{\lfloor\!\lfloor} C$

Table 3: The Failure Axioms.

$\alpha(\beta \cdot A_1 + B_1) + \alpha(\beta \cdot A_2 + B_2)$	$=$	$\alpha(\beta \cdot A_1 + \beta \cdot A_2 + B_1) + \alpha(\beta \cdot A_1 + \beta \cdot A_2 + B_2)$
$\alpha \cdot A + \alpha(B + C)$	$=$	$\alpha \cdot A + \alpha(A + B) + \alpha(B + C)$

entailed by the local store c and an arbitrary global store $\exists_x e$ iff (by the deduction theorem) $c \rightarrow d$ is entailed by $\exists_x e$, or equivalently $\forall_x(c \rightarrow d)$ is entailed by e. Note that $\forall_x(c \rightarrow d)$ is an element of $dc(\mathcal{C})$, and remember that the deduction theorem holds in $dc(\mathcal{C})$ (this is the main reason why we have intoduced the notion of $dc(\mathcal{C})$).

Next we introduce a system of axioms which characterize the specific nature of the ask and tell actions. In the following $\alpha(c)$ and $\beta(c)$ represent ask or tell actions on the constraint c. The axiom

$$\boxed{\alpha(c)(\beta(d) \cdot A + B) = \alpha(c)(\beta(c \wedge d) \cdot A + B)} \qquad (1)$$

expresses that once a constraint has been established, either by telling or asking it, it remains in the store. As a consequence, once a constraint is established, asking or telling it will have the effect of a silent transition. This is expressed by the following axiom

$$\boxed{\alpha(c)(\beta(c) \cdot A + B) = \alpha(c)(\tau \cdot A + B)} \qquad (2)$$

Table 4: τ-abstraction laws

$$\alpha \cdot \tau \cdot A = \alpha \cdot A$$

$$\tau \cdot A + B = \tau \cdot A + \tau(A + B)$$

Table 5: Quantification

$$\exists x.A = \exists_x^{true}.A$$

$$\exists_x^c.\delta = \delta$$

$$\exists_x^c.(A+B) = \exists_x^c.A + \exists_x^c.B$$

$$\exists_x^c.\mathbf{tell}(d) \cdot A = \mathbf{tell}(\exists_x(c \wedge d)) \cdot \exists_x^{c \wedge d}.A$$

$$\exists_x^c.\mathbf{ask}(d) \cdot A = \mathbf{ask}(\forall_x(c \rightarrow d)) \cdot \exists_x^c.A$$

The axioms

$$\boxed{\alpha(\mathbf{tell}(c) \cdot A + B) = \alpha(\mathbf{tell}(c) \cdot A + B) + \alpha \cdot \mathbf{tell}(c) \cdot A} \quad (3)$$

and

$$\boxed{\mathbf{tell}(c) \cdot A = \mathbf{tell}(c) \cdot A + \mathbf{ask}(d) \cdot \mathbf{tell}(c) \cdot A} \quad (4)$$

together characterize the autonomous character of a tell action, namely the fact that it can always proceed irrespective of the current store. It is worthwhile noticing the similarity of axiom 3 with the I-axiom for asynchronous communication ([7]). Axiom 4 can be informally justified as follows: suppose that the current store implies the asked constraint d, in this case the process represented by the right-hand side of the axiom can select the ask-branch, execute the tell action and proceed with A. But this behaviour can be simulated with the same observable effect by the other branch. In case the current store does not imply d, the only choice left is to execute the tell-branch. It is instructive to see why axiom 4 does not hold for ask actions: let c and d be such that neither $c \vdash d$ nor $d \vdash c$. Then the processes $A \equiv \mathbf{ask}(c) \cdot \delta$ and $B \equiv A + \mathbf{ask}(d) \cdot \mathbf{ask}(c) \cdot \delta$ can be distinguished by the context $C[\,] \equiv ([\,] + \mathbf{ask}(d) \cdot \mathbf{tell}(c) \cdot \delta) \parallel \mathbf{tell}(d) \cdot \delta$, namely after the execution of $\mathbf{tell}(d)$ the process B in $C[B]$ can select the $\mathbf{ask}(d) \cdot \mathbf{ask}(c) \cdot \delta$ branch

after which the process terminates, whereas after the execution of **tell**(d) by the process $C[A]$ the process A is suspended, and thus the enabled branch $\mathbf{ask}(d) \cdot \mathbf{tell}(c) \cdot \delta$ is selected, so formally we have $d \in \mathcal{O}[\![C[B]]\!] \setminus \mathcal{O}[\![C[A]]\!]$. However, the following axiom which allows the strengthening of an ask-guard can be shown to be valid:

$$\boxed{\mathbf{ask}(c) \cdot A = \mathbf{ask}(c) \cdot A + \mathbf{ask}(d) \cdot A} \tag{5}$$

provided $d \vdash c$. The axiom,

$$\boxed{\mathbf{tell}(c) \cdot A = \mathbf{tell}(d) \cdot \mathbf{tell}(e) \cdot A} \tag{6}$$

where $c \sim d \wedge e$, allows for the composition/decomposition of tell actions. Again, in a similar way as described above, it can be shown that a corresponding axiom for ask actions is not valid. The following restricted version of composition/decomposition,

$$\boxed{\mathbf{ask}(c) \cdot A + \mathbf{ask}(c \wedge d) \cdot B = \mathbf{ask}(c) \cdot A + \mathbf{ask}(c)(A + \mathbf{ask}(d) \cdot B)} \tag{7}$$

however, can be shown to be valid. We conclude with the following axiom

$$\boxed{\Sigma_i \alpha \Sigma_j \mathbf{ask}(c_{i_j}) \cdot A_{i_j} = \Sigma_i \alpha \Sigma_j \mathbf{ask}(c_{i_j}) \cdot A_{i_j} + \alpha \Sigma_k \mathbf{ask}(c_k) \cdot A_k} \tag{8}$$

provided for every $f \in I \to J$ if for every $k \in K \subseteq \{i_j \mid i \in I, j \in J\}$ we have $\bigwedge_i c_{i_{f(i)}} \not\vdash c_k$ then there exist i and j such that $\mathbf{ask}(\bigwedge_i c_{i_{f(i)}}) \cdot \delta \equiv \mathbf{ask}(c_{i_j}) \cdot A_{i_j}$ (Σ denotes generalized sum, and i is to be understood to range over I, j over J). This axiom can be informally justified as follows: let c be such that for no c_k we have $c \vdash c_k$. So after the execution of α the branch $\alpha \cdot \Sigma_k \mathbf{ask}(c_k) \cdot A_k$ will terminate. Now suppose that for every i there exists j such that $c \vdash c_{i_j}$. Define $f \in I \to J$ such that $c \vdash c_{i_{f(i)}}$. It follows that there exists no k such that $\bigwedge_i c_{i_{f(i)}} \vdash c_k$ (otherwise we would have $c \vdash c_k$). So there exist i and j such that $\mathbf{ask}(c_{i_j}) \cdot A_{i_j} \equiv \mathbf{ask}(\bigwedge_i c_{i_{f(i)}}) \cdot \delta$. Thus the process represented by the left-hand side of the axiom will also terminate in the current store c after the execution of α, selecting the ith branch.

Example 4.1 Consider the following equation:

$$\alpha(\mathbf{ask}(c) \cdot \delta + A) = \alpha(\mathbf{ask}(c) \cdot \delta + A) + \alpha \cdot A$$

If $A \equiv \Sigma_i \alpha_i \cdot A_i$ contains only initial ask actions, then the equation can be obtained as an instance of the axiom 8.

5 Formal justification

In this section we discuss the formal justification (i.e., soundness and completeness) of the process algebra we have presented. We indicate with the symbol $\vdash$ (not to be confused with the entailment relation $\vdash$!) the derivation

of an equality in the algebraic theory consisting of all the axioms of previous section. First we define a compositional semantics which is fully abstract with respect to $\mathcal{O}$.

In the following, $\mathcal{A}$ denotes the set of ask and tell actions. For a set S, $\mathcal{P}(S)$ is the set of all subsets of S. The domain of our semantics consists of sets of ask-tell sequences together with a constraint: formally it is given by the set $\mathcal{P}(\mathcal{A}^* \times C)$, where $\mathcal{C} = \langle C, \leq, \wedge, \mathit{true}, \mathit{false}\rangle$ is the constraint system underlying the language. Each element of this set represents a possible run of the process within an environment (context). The constraint represents the final store (as determined by the contributions of both the process and the context), final in the sense that the process cannot proceed anymore given that store. The ask-tell sequence represents the sequence of all actions performed by the process in this run.

Before describing formally the semantics we need to introduce some technical definitions. In the following, F, F_1, F_2 will indicate elements of $\mathcal{P}(\mathcal{A}^* \times C)$.

The notation $\alpha(c) \circ F$ indicates the set obtained by prefixing $\alpha(c)$ to those sequences for which c doesn't change the final result. Formally:

$$\mathbf{ask}(c) \circ F = \begin{array}{c} \{\langle \alpha(c) \cdot f, d\rangle \mid \langle f, d\rangle \in F \mathit{ and } d \vdash c\} \\ \cup \\ \{\langle \tau, d\rangle \mid \langle \epsilon, d\rangle, \langle \tau, d\rangle \in F \mathit{ and } d \vdash c\} \\ \cup \\ \{\langle \epsilon, d\rangle \mid d \not\vdash c\} \end{array}$$

and

$$\mathbf{tell}(c) \circ F = \{\langle \alpha(c) \cdot f, d\rangle \mid \langle f, d\rangle \in F \mathit{ and } d \vdash c\}$$

where ϵ denotes the empty sequence. The semantics of prefixing an ask action consists in adding the action to those sequences the final result of which entails the asked constraint. Moreover, we select the sequences $\langle \epsilon, d\rangle$ and $\langle \tau, d\rangle$, where d entails the asked constraint. With respect to these sequences, which model the situation that the process either terminates immediately or after some silent moves in the final store d, the action $\mathbf{ask}(c)$ behaves like a silent step. (Note that we additionally perform some τ-abstraction by contracting a number of silent steps into one.) Finally, we have to add those (empty) sequences consisting of a final result which does not imply the asked constraint, since in these cases the resulting process terminates immediately. Note that the main difference between the semantics of an ask and tell action is that for an ask action we need additionally to record those final stores which block the action. Since a tell action can always proceed this additional recording does not apply.

$F_1 \parallel F_2$ denotes the set of all possible interleavings of those sequences of F_1 and F_2 which result in the same final store:

$$F_1 \parallel F_2 = \{\langle f, c\rangle \mid \langle f_1, c\rangle \in F_1, \langle f_2, c\rangle \in F_2 \mathit{ and } f \in (f_1 \parallel f_2)\}$$

($f_1 \parallel f_2$ denotes the set of arbitrary interleavings of f_1 and f_2).

Finally, the local state operator is defined by

$$\exists_x^c(F) = \{\exists_x^c(\langle f, d\rangle) \mid \langle f, d\rangle \in F\},$$

where:

$$\begin{array}{lcl}\exists_x^c(\langle \mathbf{tell}(d) \cdot f, d\rangle) & = & \mathbf{tell}(\exists_x d) \circ \exists_x^{c\wedge d}(\langle f, d\rangle) \\ \exists_x^c(\langle \mathbf{ask}(d) \cdot f, d\rangle) & = & \mathbf{ask}(\forall_x(c \to d)) \circ \exists_x^c(\langle f, d\rangle) \\ \exists_x^c(\langle \epsilon, d\rangle) & = & \{\langle \epsilon, e \wedge \exists_x c\rangle \mid \ c \wedge \exists_x e \sim d\}\end{array}$$

Definition 5.1 The mapping $\mathcal{F}$ from processes to the set $\mathcal{P}(\mathcal{A}^* \times C)$ is defined as follows.

$\mathcal{F}[\![\delta]\!] = \{\langle \epsilon, c\rangle \mid \ c \in C\}$

$\mathcal{F}[\![\mathbf{ask}(c) \cdot A]\!] = \mathbf{ask}(c) \circ \mathcal{F}[\![A]\!]$

$\mathcal{F}[\![\mathbf{tell}(c) \cdot A]\!] = \mathbf{tell}(c) \circ \mathcal{F}[\![A]\!]$

$\mathcal{F}[\![A + B]\!] = (\mathcal{F}[\![A]\!] \cup \mathcal{F}[\![B]\!]) \setminus C \cup (\mathcal{F}[\![A]\!] \cap \mathcal{F}[\![B]\!] \cap C)$

$\mathcal{F}[\![A \parallel B]\!] = \mathcal{F}[\![A]\!] \parallel \mathcal{F}[\![B]\!]$

$\mathcal{F}[\![\exists x.A]\!] = \exists_x^{true}(\mathcal{F}[\![A]\!])$.

The set of all possible final results of the process δ is the set C itself, since δ does not impose any constraints and all constraints are final for it.

The semantics of a process $A+B$ consists of the non-empty (with respect to the actions) sequences of A and B, plus those empty sequences the final result of which belongs both to A and B. These latter sequences represent those stores from which neither A nor B can proceed. Here C is used as an abbreviaton for $\{\epsilon\} \times C$.

Quantification is described in terms of the state operator $\exists_x^c$ where c represents the local store which contains information about the local x.

The correctness of $\mathcal{F}$ with respect to $\mathcal{O}$ is stated in the following theorem:

Theorem 5.2 *For every process A we have*

$$\begin{array}{ll}\mathcal{O}[\![A]\!] = \{c \mid & \exists f.\ \langle f, c\rangle \in \mathcal{F}[\![A]\!], con(f) \sim c, \\ & \forall f'(\ f' \cdot \mathbf{ask}(d) \preceq f \Rightarrow con(f') \vdash d)\}\end{array}$$

Here $con(f)$ denotes the conjunction of all the constraints occurring of the ask and tell actions of f, and $\preceq$ denotes the prefix relation.

However $\mathcal{F}$ is not fully abstract with respect to $\mathcal{O}$. We need the following closure conditions which characterize the monotonic nature of the computational model:

Definition 5.3 For $F \in \mathcal{P}(\mathcal{A}^* \times C)$ let $Sat(F)$ denote the smallest set containing F which is closed under the following conditions:

C1 $f \cdot \alpha(c) \cdot \beta(d) \cdot f' \in F \;\Rightarrow\; f \cdot \alpha(c) \cdot \beta(c \wedge d) \cdot f'$

C2 $f \cdot \alpha(c) \cdot \beta(c) \cdot f' \in F \;\Leftrightarrow\; f \cdot \alpha(c) \cdot \tau \cdot f'$

C3 $f \cdot \mathbf{tell}(c) \cdot \mathbf{tell}(d) \cdot f' \in F \;\Leftrightarrow\; f \cdot \mathbf{tell}(c \wedge d) \cdot f' \in F$

C4 $f \cdot \mathbf{ask}(c) \cdot \mathbf{ask}(d) \cdot f' \in F \;\Rightarrow\; f \cdot \mathbf{ask}(c \wedge d) \cdot f' \in F$

C5 $f \cdot \mathbf{ask}(c) \cdot f' \in F \;\Rightarrow\; f \cdot \mathbf{ask}(d) \cdot f' \in F \; (d \vdash c)$

C6 $f \cdot \mathbf{ask}(c \wedge d) \cdot f' \in F,\; f \cdot \mathbf{ask}(c) \cdot f'' \in F \;\Rightarrow\; f \cdot \mathbf{ask}(c) \cdot \mathbf{ask}(d) \cdot f' \in F$

C7 $f \cdot \alpha \cdot \tau \cdot f' \in F \;\Leftrightarrow\; f \cdot \alpha \cdot f' \in F$

(Note that we both use f to denote an element of $\mathcal{A}^*$ and $\mathcal{P}(\mathcal{A}^* \times C)$.) In C5 it is assumed that the constraint d is entailed by the final result.

In the full paper we show how these conditions can be expressed by the axioms.

Next we introduce the semantics $\mathcal{F}'$:

Definition 5.4 For every process A we define $\mathcal{F}'[\![A]\!] = Sat(\mathcal{F}[\![A]\!])$.

In the full paper we show that $\mathcal{F}'$ is compositional and fully abstract with respect to $\mathcal{O}$, which together with the correctness, gives the following theorem:

Theorem 5.5 *For any processes A and B we have*

$$A \doteq B \Leftrightarrow \mathcal{F}'[\![A]\!] = \mathcal{F}'[\![B]\!].$$

Given this characterization of the congruence we can prove the soundness and completeness of the axiom system:

Theorem 5.6 *For any processes A and B*

$$\vdash A = B \;\Leftrightarrow\; A \doteq B$$

Proof-sketch By Theorem 5.5 to prove soundness it suffices to show for any axiom $A = B$ that $\mathcal{F}'[\![A]\!] = \mathcal{F}'[\![B]\!]$. For a detailed proof of the completeness we refer to the full paper. The structure of the proof consists of a completeness result for basic processes, i.e., processes which are built up from the ask/tell primitives and δ using prefixing and choice only, and an elemination theorem which states that every process is provable equal to a basic process. In the completeness result for basic processes the following expressiveness of the closure conditions plays a crucial role: let $\mathcal{F}^0[\![A]\!] = \mathcal{F}[\![A]\!]$ and $\mathcal{F}^{n+1}[\![A]\!] = Sat(\mathcal{F}^n[\![A]\!])$, then for every n and for every basic process A there exists a basic process A_n such that $\vdash A = A_n$ and $\mathcal{F}[\![A_n]\!] = \mathcal{F}^n[\![A]\!]$. □

6 Future Research

We investigated an algebraic axiomatization of concurrent constraint programming. An essential feature of our computational model is that the execution of a tell action is not constrained by consistency requirements; it is modelled as an autonomous action. Also with respect to an ask action we do not require the asked constraint to be consistent with the current store, we only require the asked constraint to be entailed by it. It would be interesting to study algebraically other models which do impose consistency requirements on the execution of a ask/tell action. These other models then would require additionally an algebraic theory for inconsistency or failure.

References

[1] S.D. Brookes, C.A.R. Hoare, and W. Roscoe. *A theory of communicating sequential processes.* Journal of ACM, 31:499–560, 1984.

[2] J.A. Bergstra and J.W. Klop. *Process algebra: specification and verification in bisimulation semantics.* Mathematics and Computer Science II, CWI Monographs, pages 61 - 94. North-Holland, 1986.

[3] J.A. Bergstra, J.W. Klop, and E.-R. Olderog. *Readies and failures in the algebra of communicating processes.* SIAM J. on Computing, 17(6):1134 - 1177, 1988.

[4] J.A. Bergstra, J.W. Klop, and J.V. Tucker. *Process algebra with asynchronous communication mechanisms.* S.D. Brookes, A.W. Roscoe, and G. Winskel, editors, Proc. Seminar on Concurrency, volume 197 of Lecture Notes in Computer Science, pages 76 - 95. Springer-Verlag, 1985.

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

[6] K.L. Clark and S. Gregory. *PARLOG: parallel programming in logic.* ACM Trans. on Programming Languages and Systems, (8):1–49, 1986.

[7] F.S. de Boer, J.W. Klop, and C. Palamidessi. *Asynchronous communication in process algebra.* Proc. of LICS 92, IEEE Computer Society Press, 1992. To appear.

[8] L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras (Part I).* North-Holland, 1971.

[9] R. Milner. *A Calculus of Communicating Systems*, volume 92 of Lecture Notes in Computer Science. Springer-Verlag, New York, 1980.

[10] V.A. Saraswat. *Concurrent Constraint Programming Languages.* PhD thesis, Carnegie-Mellon University, January 1989. Published by The MIT Press, U.S.A., 1990.

[11] D. Scott. *Domains for denotational semantics.* Proc. of ICALP, 1982.

[12] E.Y. Shapiro. *A subset of Concurrent Prolog and its interpreter.* Technical Report TR-003, Institute for New Generation Computer Technology (ICOT), Tokyo, 1983.

[13] E. Y. Shapiro. *Concurrent Prolog: A progress report.* Computer, 19(8):44–58, 1986.

[14] V.A. Saraswat and M. Rinard. *Concurrent constraint programming.* Proc. of the seventeenth ACM Symposium on Principles of Programming Languages, pages 232–245. ACM, New York, 1990.

[15] V.A. Saraswat, M. Rinard, and P. Panangaden. *Semantics foundations of Concurrent Constraint Programming.* Proc. of the eighteenth ACM Symposium on Principles of Programming Languages. ACM, New York, 1991.

[16] K. Ueda. *Guarded Horn Clauses.* E. Y. Shapiro, editor, Concurrent Prolog: Collected Papers. The MIT Press, 1987.

[17] K. Ueda. *Guarded Horn Clauses, a parallel logic programming language with the concept of a guard.* M. Nivat and K. Fuchi, editors, Programming of Future Generation Computers, pages 441–456. North Holland, Amsterdam, 1988.

In this series appeared:

91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of P-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.
91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.

91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.
91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.
91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.

92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.
92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	Are Types for Natural Language? P. 32.
92/21	F.Kamareddine	Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.

92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.

92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.

93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.