Department of Philosophy - Utrecht University

Invariants in Process Algebra with Data

M.A. Bezem
J.F. Groote

Logic Group
Preprint Series
No. 98
September 1993

Department of Philosophy
Utrecht University
Heidelberglaan 8
3584 CS Utrecht
The Netherlands

©1993, Department of Philosophy - Utrecht University
ISBN 90-393-0179-4
ISSN 0929-0710
Dr. A. Visser, Editor

Invariants in Process Algebra with Data

Marc Bezem
Jan Friso Groote
Department of Philosophy, Utrecht University
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands
Email: Marc.Bezem@phil.ruu.nl, JanFriso.Groote@phil.ruu.nl

Abstract

We provide rules for calculating with invariants in process algebra with data, and illustrate these with examples. The new rules turn out to be equivalent to the well known Recursive Specification Principle which states that guarded recursive equations have at most one solution. In the setting with data this is reformulated as 'every *convergent* linear process operator has at most one fixed point' (CL-RSP). As a consequence, one can carry out verifications in well-known process algebras satisfying CL-RSP using invariants.

Key Words & Phrases: process algebra, invariants, data types.
1985 Mathematics Subject Classification: 68B10.
1987 CR Categories: D.2.4, D.3.1, F.3.1.
Note: The authors are partly supported by the Netherlands Computer Science Research Foundation (SION) with financial support of the Netherlands Organization for Scientific Research (NWO).

1 Introduction

Process algebra allows one to prove that certain processes are equal; typically, an abstraction of a (generally complicated) implementation and a (preferably compact) specification of the external behaviour of the same system are shown equal. Process algebra has been used to verify a variety of distributed systems such as protocols, systolic arrays and computer integrated manufacturing systems (see e.g. [1]). The main technique to prove such process identities consists of algebraic manipulation with terms, i.e. elementary calculations, a technique that has been proved successful in mathematics.

However, current process algebraic techniques are considered unsuitable to verify large and complex systems. In particular, it appears that knowledge and intuition about the correct functioning of distributed systems can hardly be explicitly incorporated in the calculations. Since the algebraic proofs are usually long, many attempts to construct a proof smother into a large amount of undirected calculations.

Therefore, the need is felt to adapt the theory of process algebra in such a way that intuitions about the functioning of distributed systems can be expressed in a natural way and can be used in calculations. One of the most successful techniques in this respect is the use of invariants, as put forward prominently by Dijkstra and Hoare. Almost all existing verifications of sequential and distributed programs use relations between variables that remain

valid during the course of the program. A comprehensive survey for distributed programs is [4]. The closest approximation of the techniques in this paper we could find in the classical literature is [10].

In process algebra — as far as we know — invariants are seldom explicitly used. An early but incomplete attempt to verify a sliding window protocol in process algebra using invariants is [6]. A Hoare Logic based approach to invariants in process algebra has been elaborated in [8, 11]. Rudimentary, implicit use of invariants in proces algebra, however, is quite common. For example, the set of accessible states of a process is an invariant of that process (modulo the identification of subsets of the state space with relations between the variables).

We investigate invariants in the setting of μCRL [7, 9], in which process algebra is combined with data. It is straightforward to formulate invariants in μCRL as predicates. We first adapt the Recursive Specification Principle (RSP), which states that guarded recursive specifications have unique solutions, such that it can effectively be used in a setting with data. We introduce convergent linear process operators (CLPO's) and formulate the principle CL-RSP, which states that every convergent linear recursive operator has at most one fixed point. Then we provide several formulations of equivalent principles to allow smooth calculations using invariants. We illustrate almost all rules with examples, and provide two somewhat larger examples at the end. Remarkably, all new rules turn out to be equivalent to CL-RSP. As a consequence, one can carry out verifications using invariants in well-known process algebras satisfying CL-RSP.

The new rules for invariants have been applied to different sizable examples (see e.g. [3]) and their use indeed shows that the intuitions about the functioning of distributed systems is reflected more clearly in the proofs. These proofs also suggest that the verification of distributed systems, formerly out of reach, is now plausible.

Acknowledgements. We thank Jan Bergstra, Wim Hesselink, Tonny Hurkens, Jaco van de Pol, Alex Sellink, Jan Springintveld and Jos van Wamel for discussions on the topic of this paper and comments on an early draft.

2 Preliminaries

We assume the existence of non-empty, disjoint sets with data elements, which are called *data types* and are denoted by letters D and E. Furthermore, we assume a set of many sorted operations on these sets, which are called *data operations*. There is one standard data type **Bool**, which consists of the elements t and f. We assume that the standard boolean operations are defined on **Bool**. We also assume the existence of a set *pAct* that contains parameterised atomic actions. Every element of *pAct* comes equipped with the data type of its parameter. The elements of *pAct* are regarded as mappings from data types to processes.

Definition 2.1. A *pCRL*^{1}*-algebra* is a set $\mathbb{P}$, disjoint from the data types, with operations

$a : D \to \mathbb{P}$ (for all $a \in pAct$, D the data type of a)
$\delta, \tau : \mathbb{P}$
$+, \cdot : \mathbb{P} \times \mathbb{P} \to \mathbb{P}$
$\Sigma : (D \to \mathbb{P}) \to \mathbb{P}$ (for each data type D)
$\_ \lhd \_ \rhd \_ : \mathbb{P} \times \mathbf{Bool} \times \mathbb{P} \to \mathbb{P}$

A1	$x+y=y+x$	SUM1	$\Sigma_{d:D}x=x$
A2	$x+(y+z)=(x+y)+z$	SUM3	$\Sigma p=\Sigma p+p(d)$
A3	$x+x=x$	SUM4	$\Sigma_{d:D}(p(d)+q(d))=\Sigma p+\Sigma q$
A4	$(x+y)\cdot z=x\cdot z+y\cdot z$	SUM5	$\Sigma_{d:D}(p(d)\cdot x)=(\Sigma p)\cdot x$
A5	$(x\cdot y)\cdot z=x\cdot(y\cdot z)$	SUM11	$(\forall d\in D\ p(d)=q(d))\rightarrow\Sigma p=\Sigma q$
A6	$x+\delta=x$		
A7	$\delta\cdot x=\delta$	Bool1	$\neg(\mathsf{t}=\mathsf{f})$
		Bool2	$\neg(b=\mathsf{t})\rightarrow b=\mathsf{f}$
B1	$c\cdot\tau=c$	C1	$x\lhd\mathsf{t}\rhd y=x$
B2	$c\cdot(\tau\cdot(x+y)+x)=c\cdot(x+y)$	C2	$x\lhd\mathsf{f}\rhd y=y$

Table 1: $x,y,z\in\mathbb{P}$, c is τ or $a(d)$ for $a\in pAct$, $d\in D$ and $p,q:D\rightarrow\mathbb{P}$, $b\in\mathbf{Bool}$

satisfying the (conditional) equations A1–7, SUM1,3,4,5,11, Bool1,2, C1,2 from Table 1. If the algebra also satisfies the equations B1 and B2 for branching bisimulation [5] it is called a *$pCRL^{1\tau}$-algebra.*

The name pCRL stands for a pico Common Representation Language. The superscript 1 refers to the fact that, contrary to μCRL, all actions have exactly one parameter. By using the data type with one element and by using pairing, actions depending on zero or more than one data type can be simulated. Therefore, we use zero or more than one parameter whenever convenient.

Table 1 specifies a subset of the axioms from the proof theory of μCRL [9]. The operations $+$ and $\cdot$ and equations A1–7 and B1,2 are standard for process algebra (see [2]) and therefore not explained. The operation $\_\lhd\_\rhd\_$ is the *then-if-else* operation. The sum operation Σ over a data type D expresses that its argument $p:D\rightarrow\mathbb{P}$ may be executed with some data element d from D. Instead of $\Sigma(\lambda d{:}D.x)$ we generally write $\Sigma_{d:D}x$. Note that we use explicitly typed λ notation to denote mappings. If convenient we sometimes drop the explicit types. We also use a meta-sum notation $\Sigma_{i\in I}p_i$ for $p_1+p_2+\cdots+p_n$ when $I=\{1,\ldots,n\}$. The difference between the notations is the use of the colon versus the use of the membership symbol. We use the convention that $\cdot$ binds stronger than Σ, followed by $\_\lhd\_\rhd\_$, and $+$ binds weakest.

We call a mapping $p:D\rightarrow\mathbb{P}$ a *parameterised process* and a mapping $(D\rightarrow\mathbb{P})\rightarrow(E\rightarrow\mathbb{P})$ a *process transformation* (*process operator* if $D=E$). Generally, letters p,q are used for parameterised processes and letters Φ,Ψ,Ξ (Θ) refer to process operators (transformations).

Definition 2.2. A process operator Ψ is called (pCRL-)*expressible* iff Ψ can be written as

$$\lambda p{:}D\rightarrow\mathbb{P}.\lambda d{:}D.t(p,d)$$

where $t(p,d)$ only consists of operations mentioned in Definition 2.1.

A process operator Ψ is called *linear* iff Ψ can be written as

$$\lambda p{:}D\rightarrow\mathbb{P}.\lambda d{:}D.\Sigma_{i\in I}\Sigma_{e_i:D_i}c_i(f_i(d,e_i))\cdot p(g_i(d,e_i))\lhd b_i(d,e_i)\rhd\delta+\Sigma_{i\in I'}\Sigma_{e_i:D'_i}c'_i(f'_i(d,e_i))\lhd b'_i(d,e_i)\rhd\delta$$

for some finite index sets I, I', actions $c_i, c'_i \in pAct \cup \{\tau\}$, data types D_i, D'_i, D_{c_i} and $D_{c'_i}$, functions $f_i : D \to D_i \to D_{c_i}$, $g_i : D \to D_i \to D$, $b_i : D \to D'_i \to \mathbf{Bool}$, $f'_i : D \to D'_i \to D_{c'_i}$, $b'_i : D \to D'_i \to \mathbf{Bool}$.

Definition 2.3. A linear process operator (*LPO*) Ψ written in the form above is called *convergent* iff there is a well-founded ordering $<$ on D such that $g_i(d, e_i) < d$ for all $d \in D$, $i \in I$ and $e_i \in D_i$ with $c_i = \tau$ and $b_i(d, e_i)$.

In this paper we restrict ourselves to *linear* process operators, thus excluding, for example, $\lambda pd.\, a{\cdot}b{\cdot}p(d)$. Note that the notion of a convergent linear process operator both specialises and generalises usual notions of a guarded recursive specification. First, $\lambda pd.\, a{\cdot}b{\cdot}p(d)$ is not a CLPO, but would be considered guarded. Second unguarded occurrences are allowed in a CLPO on the condition that the parameter decreases in the sense of some well-founded ordering. The latter seems an unnecessary complication, but is in fact crucial for applications such as in Subsection 4.3 and in [3]. We use convergent LPO's for the same reason as guarded recursive specifications are used, namely to uniquely determine (parameterised) processes. For this purpose, the *Recursive Definition Principle* (RDP) and the *Recursive Specification Principle* (RSP) were introduced in [2]. We reformulate these principles in the presence of data.

Definition 2.4. A pCRL-algebra is said to satisfy *RDP* iff every expressible process operator Ψ has at least one fixed point, i.e. there exists a $p : D \to \mathbb{P}$ such that $p = \Psi p$.

A pCRL-algebra is said to satisfy *CL-RSP* iff every convergent linear process operator (CLPO) has at most one fixed point.

In the sequel we assume that all algebras that we consider satisfy CL-RSP. RDP is only used in the examples, where it will be tacitly assumed, too.

If RDP and CL-RSP hold, we can use CLPO's to define (parameterised) processes. Generally, and conforming to μCRL [7] this is denoted as follows. If $\Psi : (D \to \mathbb{P}) \to (D \to \mathbb{P})$ is a CLPO, we write

proc $p(x{:}D) = \Psi p x$

This means that we define $p : D \to \mathbb{P}$ as the unique fixed point of Ψ.

Example 2.5. Consider the following two processes. We assume that the pCRL-algebra contains the data type of natural numbers $\mathbb{N}$ with the standard operations, as well as a data type $\mathbb{N} \times \mathbb{N}$ with pairing and projection operations. Here and below we simply write $q(m, n{:}\mathbb{N})$ to conform to standard notations, where actually $q(o : \mathbb{N} \times \mathbb{N})$ would be required, with m and n the first and, respectively, second projection of o.

proc $p(n{:}\mathbb{N}) = a(n){\cdot}p(n) + \tau{\cdot}p(n-1) \lhd n > 0 \rhd \delta$
$q(m, n{:}\mathbb{N}) = (\Sigma_{n':\mathbb{N}} \tau{\cdot}q(m-1, n') \lhd n = 0 \rhd \tau{\cdot}q(m, n-1)) \lhd m > 0 \rhd \tau$

It is obvious that the first process definition is convergent, using the standard ordering on natural numbers. In the second example, the right hand side should first be linearised into

$(\tau \lhd m = 0 \rhd \delta) + (\tau{\cdot}q(m, n-1) \lhd m > 0 \wedge n > 0 \rhd \delta) +$
$(\Sigma_{n':\mathbb{N}} \tau{\cdot}q(m-1, n') \lhd m > 0 \wedge n = 0 \rhd \delta)$

Taking the lexicographical ordering on $\mathbb{N} \times \mathbb{N}$ defined by $\langle m, n \rangle < \langle m', n' \rangle$ iff $(m < m') \vee (m = m' \wedge n < n')$, one easily sees that the second process definition is convergent. With CL-RSP it follows that both p and q are unique parameterised processes. (In the second example this can also be established by proving by transfinite induction that $q(m, n) = \tau$.)

3 Invariants

In this section we provide a number of equivalent versions of CL-RSP. Some of these are formulated to accommodate the convenient use of invariants. Due to the abstract setting of this section, the true content of the lemmas may be hard to grasp, although the proofs are very short and straightforward. Therefore, almost all lemmas are illustrated with examples.

Lemma 3.1. *(Symmetrical Lemma). Assume that the following diagram commutes, i.e.* $\Theta(\Phi p) = \Psi(\Theta p)$ *for all* p *and* $\Theta'(\Phi' p') = \Psi(\Theta' p')$ *for all* p'.

$$\begin{array}{ccc} D{\to}\mathbb{P} & \xrightarrow{\Phi} & D{\to}\mathbb{P} \\ \Theta \downarrow & & \Theta \downarrow \\ E{\to}\mathbb{P} & \xrightarrow{\Psi} & E{\to}\mathbb{P} \\ \Theta' \uparrow & & \Theta' \uparrow \\ D'{\to}\mathbb{P} & \xrightarrow{\Phi'} & D'{\to}\mathbb{P} \end{array}$$

Let Ψ *be a CLPO. If* $\Theta p = \Theta(\Phi p)$ *and* $\Theta' p' = \Theta'(\Phi' p')$, *then* $\Theta p = \Theta' p'$. *In particular, if* $p = \Phi p$ *and* $p' = \Phi' p'$ *then* $\Theta p = \Theta' p'$.

Proof.

$$\left.\begin{array}{l} \Theta p = \Theta(\Phi p) = \Psi(\Theta p) \\ \Theta' p' = \Theta'(\Phi' p') = \Psi(\Theta' p') \end{array}\right\} \overset{\text{CL-RSP}}{\Longrightarrow} \Theta p = \Theta' p'.$$

□

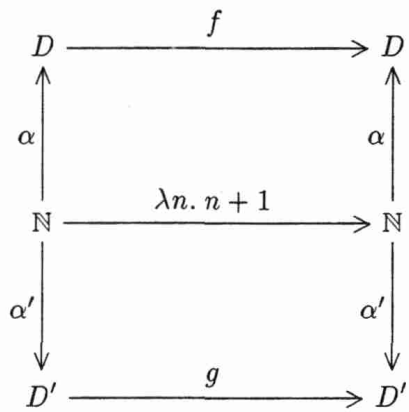
An important special case of this lemma is $\Theta = \lambda p.\, p \circ \alpha$ for some $\alpha : E{\to}D$, where $\circ$ denotes function composition. Commutativity of the upper part of the diagram then boils down to $(\Phi p) \circ \alpha = \Psi(p \circ \alpha)$ for all p. Spelling this out leads to the well-known notion of *bisimulation mapping* α, as becomes apparent in the next example.

It should be noted that the proof above does not depend on the particular structure of CLPO's. The existence of unique fixed points is the only fact that is used in the proofs. Therefore the result in Lemma 3.1 and all subsequent lemmas carry over to all process operators assuming that they have unique fixed points.

Example 3.2. Consider the following two processes:

proc $q(x{:}D) = a(d(x))\cdot q(f(x))$
$q'(y{:}D') = a(e(y))\cdot q'(g(y))$

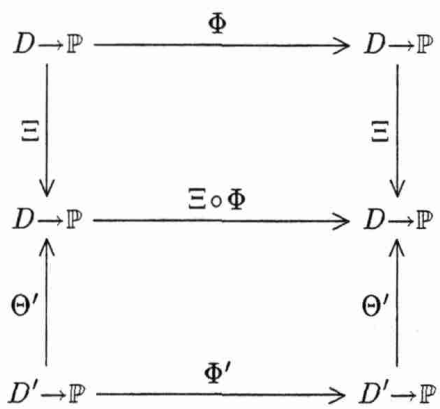
We prove by the Symmetrical Lemma that $q(x_0) = q'(y_0)$, provided $d(f^n(x_0)) = e(g^n(y_0))$ for all $n \in \mathbb{N}$. Take $E = \mathbb{N}$. Define $\Phi : (D{\to}\mathbb{P}){\to}(D{\to}\mathbb{P})$ by $\Phi p = \lambda x.\, a(d(x))\cdot p(f(x))$, $\Phi' : (D'{\to}\mathbb{P}){\to}(D'{\to}\mathbb{P})$ by $\Phi' p' = \lambda y.\, a(e(y))\cdot p'(g(y))$, and $\Psi : (\mathbb{N}{\to}\mathbb{P}){\to}(\mathbb{N}{\to}\mathbb{P})$ by $\Psi r = \lambda n.a(d(f^n(x_0)))\, r(n{+}1)$. Then we have that Ψ is convergent and $q = \Phi q$, $q' = \Phi' q'$. Moreover, define bisimulation mappings $\alpha : \mathbb{N}{\to}D$, $\alpha' : \mathbb{N}{\to}D'$ by putting $\alpha n = f^n(x_0)$, $\alpha' n = g^n(y_0)$. Then the following diagram commutes.



Since $d(f^n(x_0)) = e(g^n(y_0))$ for all $n \in \mathbb{N}$, α and α' are indeed bisimulation mappings. Define process transformations $\Theta : (D{\to}\mathbb{P}){\to}(\mathbb{N}{\to}\mathbb{P})$, $\Theta' : (D'{\to}\mathbb{P}){\to}(\mathbb{N}{\to}\mathbb{P})$ by $\Theta p = p \circ \alpha$, $\Theta' p' = p' \circ \alpha'$. Then it follows from the previous diagram that the diagram in the Symmetrical Lemma commutes. Hence $\Theta q = \Theta' q'$ and $q(x_0) = q'(y_0)$.

The following lemma facilitates calculating with invariants. A typical application is with $\Xi = \lambda px.\, p(x) \lhd I(x) \rhd \delta$, where $I(x)$ is an invariant of $\Phi : (D{\to}\mathbb{P}){\to}(D{\to}\mathbb{P})$. We explain the intuition later.

Lemma 3.3. *(Asymmetrical Lemma). Assume that the following diagram commutes.*



Assume also that $\Xi \circ \Phi$ is a CLPO. If $\Xi p = \Xi(\Phi p)$ and $\Theta' p' = \Theta'(\Phi' p')$ then $\Xi p = \Theta' p'$. In particular, if $p = \Phi p$ and $p' = \Phi' p'$ then $\Xi p = \Theta' p'$.

Proof. Take $\Psi = \Xi \circ \Phi$ in Lemma 3.1. □

Example 3.4. Consider the following two processes.

$$\begin{aligned}\textbf{proc}\quad & q(x{:}\mathbb{N}) = a(even(x))\cdot q(x+2)\\ & q' = a(\mathbf{t})\cdot q'\end{aligned}$$

We prove by the Asymmetrical Lemma that $even(n) \rightarrow q(n) = q'$. As stated in the preliminaries, q' above is shorthand for $q'(y)$ with y a variable of a type $\mathbb{1}$ containing only one element. Thus the second equation actually reads:

$$q'(y{:}\mathbb{1}) = a(\mathbf{t})\cdot q'(y)$$

Take $D = \mathbb{N}$, $D' = \mathbb{1}$. Define $\Phi : (\mathbb{N}\rightarrow\mathbb{P})\rightarrow(\mathbb{N}\rightarrow\mathbb{P})$ by $\Phi p = \lambda n.\ a(even(n))\cdot p(n+2)$, and $\Phi' : (\mathbb{1}\rightarrow\mathbb{P})\rightarrow(\mathbb{1}\rightarrow\mathbb{P})$ by $\Phi' p' = \lambda y.\ a(\mathbf{t})\cdot p'(y)$. Moreover, define $\Xi : (\mathbb{N}\rightarrow\mathbb{P})\rightarrow(\mathbb{N}\rightarrow\mathbb{P})$ by $\Xi p = \lambda n.\ p(n) \lhd even(n) \rhd \delta$ and, for given $y \in \mathbb{1}$, $\Theta' : (\mathbb{1}\rightarrow\mathbb{P})\rightarrow(\mathbb{N}\rightarrow\mathbb{P})$ by $\Theta' p' = \lambda n.\ p'(y) \lhd even(n) \rhd \delta$. Then we have that $\Xi \circ \Phi$ is convergent and $q = \Phi q$, $q' = \Phi' q'$. We verify the commutativity of the diagram. The lower part of the diagram requires that we prove the equation $(\Xi(\Phi(\Theta' p')))n = (\Theta'(\Phi' p'))n$. The LHS evaluates to $(a(even(n))\cdot(p'(y) \lhd even(n+2) \rhd \delta)) \lhd even(n) \rhd \delta$ and the RHS to $(a(\mathbf{t})\cdot p'(y)) \lhd even(n) \rhd \delta$. These are equal since $even(n) \rightarrow (a(even(n)) = a(\mathbf{t}) \wedge even(n+2))$. The upper part of the diagram requires proving the equation $(\Xi(\Phi(\Xi p)))n = (\Xi(\Phi p))n$. This equation is handled in a similar way, using $even(n) \rightarrow even(n+2)$ (in other words: $even(n)$ is an invariant of Φ). It follows by the Asymmetrical Lemma that $\Xi q = \Theta' q'$, i.e. $\lambda n.\, q(n) \lhd even(n) \rhd \delta = \lambda n.\, q'(y) \lhd even(n) \rhd \delta$, so $even(n) \rightarrow q(n) = q'(y)$.

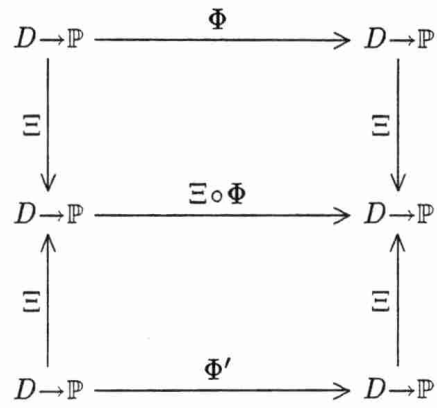
Intuition. The following may help to elucidate the relation between an invariant and the commutativity of the upper part of the diagram in the Asymmetrical Lemma. Let $\Xi = \lambda p x.\ p(x) \lhd I(x) \rhd \delta$, where $I(x)$ is an invariant of $\Phi : (D\rightarrow\mathbb{P})\rightarrow(D\rightarrow\mathbb{P})$. Expanding the equation $\Xi(\Phi(\Xi p)) = \Xi(\Phi p)$, which corresponds to the upper part of the diagram, yields the following equation:

$$\lambda x.\, \Phi(\lambda x'.\, p(x') \lhd I(x') \rhd \delta)x \lhd I(x) \rhd \delta = \lambda x.\, \Phi p x \lhd I(x) \rhd \delta$$

Typically, Φ does a number of calls of the form $p(x')$. These are the so-called recursive calls when Φ is regarded as defining the process p by $p = \Phi p$. The equation expresses that at such a point the invariant must be valid, i.e. p in the RHS may be replaced by $\lambda x'.p(x) \lhd I(x) \rhd \delta$. The displayed equation is clearly satisfied when $I(x) \rightarrow I(x')$ for every x, x' such that $p(x')$ occurs in $\Phi p x$. This is exactly the case for an invariant I.

The following trivial consequence of the previous lemmas is useful in simplifying recursive processes using invariants.

Lemma 3.5. *(Simplifying Lemma). Assume that the following diagram commutes.*



Assume that $\Xi\circ\Phi$ is a CLPO. If $\Xi p = \Xi(\Phi p)$ and $\Xi p' = \Xi(\Phi' p')$ then $\Xi p = \Xi p'$. In particular, if $p = \Phi p$ and $p' = \Phi' p'$ then $\Xi p = \Xi p'$.

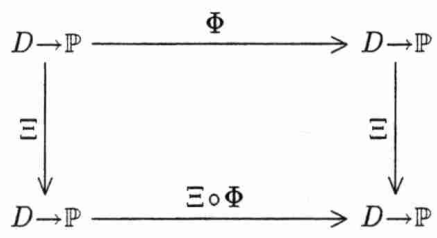
Proof. Apply Lemma 3.3 with $\Theta' = \Xi$. □

Example 3.6. Consider the following two processes.

proc $q(x{:}\mathbb{N}) = a(x, even(x))\cdot q(x+2)$
$q'(x{:}\mathbb{N}) = a(x, \mathbf{t})\cdot q'(x+2)$

We prove by the Simplifying Lemma that $even(n) \rightarrow q(n) = q'(n)$. Take $D = \mathbb{N}$ and define $\Phi : (\mathbb{N}\to\mathbb{P})\to(\mathbb{N}\to\mathbb{P})$ by $\Phi p = \lambda n.\, a(n, even(n))\cdot p(n+2)$, and $\Phi' : (\mathbb{N}\to\mathbb{P})\to(\mathbb{N}\to\mathbb{P})$ by $\Phi' p = \lambda n.\, a(n, \mathbf{t})\cdot p(n+2)$. Moreover, define $\Xi : (\mathbb{N}\to\mathbb{P})\to(\mathbb{N}\to\mathbb{P})$ by $\Xi p = \lambda n.\, p(n) \lhd even(n) \rhd \delta$. Then we have that $\Xi\circ\Phi$ is convergent and $q = \Phi q$, $q' = \Phi' q'$. The commutativity of the diagram is easily proved in a way similar to Example 3.4. It follows by the Simplifying Lemma that $\Xi q = \Xi q'$, i.e. $\lambda n.\, q(n) \lhd even(n) \rhd \delta = \lambda n.\, q'(n) \lhd even(n) \rhd \delta$, so $even(n) \rightarrow q(n) = q'(n)$.

Lemma 3.7. *(Invariant Lemma). Assume that the following diagram commutes.*



Assume also that $\Xi\circ\Phi$ is a CLPO. If $\Xi p = \Xi(\Phi p)$ and $\Xi p' = \Xi(\Phi p')$ then $\Xi p = \Xi p'$.

Proof. Apply Lemma 3.5 with $\Phi' = \Phi$. □

Corollary 3.8 *(Abstract Invariant Corollary). Assume that the following diagram commutes.*

$$\begin{array}{ccc} D{\to}\mathbb{P} & \xrightarrow{\quad\Phi\quad} & D{\to}\mathbb{P} \\ \Big\downarrow{\scriptstyle\Xi} & & \Big\downarrow{\scriptstyle\Xi} \\ D{\to}\mathbb{P} & \xrightarrow{\quad\Xi\circ\Phi\quad} & D{\to}\mathbb{P} \end{array}$$

Assume that Ξ is defined by an invariant I of Φ, i.e. $\Xi = \lambda px.\, p(x) \lhd I(x) \rhd \delta$, and that $\Xi\circ\Phi$ is a CLPO. Then for all $q, q'{:}D{\to}\mathbb{P}$ such that $I(x) \to q(x) = \Phi qx$ and $I(x) \to q'(x) = \Phi q'x$ we have $I(x) \to q(x) = q'(x)$.

Proof. Note that $I(x) \to q(x) = \Phi qx$ is equivalent to $\Xi q = \Xi(\Phi q)$, and likewise for q'. □

Corollary 3.9 *(Concrete Invariant Corollary). Assume*

$$\Phi = \lambda p{:}D{\to}\mathbb{P}.\lambda d{:}D.\, \Sigma_{j\in J}\Sigma_{e_j:D_j} c_j(f_j(d,e_j))\cdot p(g_j(d,e_j)) \lhd b_j(d,e_j) \rhd \delta + \Sigma_{j\in J'}\Sigma_{e_j:D'_j} c'_j(f'_j(d,e_j)) \lhd b'_j(d,e_j) \rhd \delta$$

*is a LPO. If for some predicate $I : D{\to}$**Bool***

$\lambda pd.\Phi pd \lhd I(d) \rhd \delta$ is convergent, and
$I(d) \wedge b_j(d,e_j) \to I(g_j(d,e_j))$ for all $j \in J$, $d \in D$ and $e_j \in D_j$,

i.e. I is an invariant of Φ, and for some $q : D{\to}\mathbb{P}$, $q' : D{\to}\mathbb{P}$ we have

$I(d) \to q(d) = \Phi qd$,
$I(d) \to q'(d) = \Phi q'd$,

then

$I(d) \to q(d) = q'(d)$.

Proof. We apply the Abstract Invariant Corollary. Let the conditions be as above. Define $\Xi = \lambda pd.p(d) \lhd I(d) \rhd \delta$. By assumption $\Xi\circ\Phi$ is a CLPO. According to the intuition after Example 3.4 we can show that the diagram in Corollary 3.8 commutes. As by assumption $I(d) \to q(d) = \Phi qd$ and $I(d) \to q'(d) = \Phi q'd$, we may conclude that $I(d) \to q(d) = q'(d)$. □

Proposition 3.10. Lemmas 3.1, 3.3, 3.5, 3.7, Corollaries 3.8 and 3.9 are equivalent to CL-RSP.

Proof. Consider the following cycle of implications: CL-RSP → Lemma 3.1 → Lemma 3.3 → Lemma 3.5 → Lemma 3.7 → Corollary 3.8 → Corollary 3.9 → CL-RSP. All implications but the last have been established already. The last implication is obvious: take $I = \lambda x.\mathrm{t}$ in the Concrete Invariant Corollary 3.9. □

4 Two larger examples

We provide two examples that show how the results above can be applied. In the first example two queues are shown equal and in the second it is shown how two persons can play table tennis.

4.1 Two queues

Example 4.1. Consider the following two queues:

$$\begin{aligned}\textbf{proc}\quad & q(old, new{:}\mathbb{N}, v{:}set, n{:}\mathbb{N}) = \\ & \quad s(get(v, old))\cdot q(old+1, new, v - \langle old, get(v, old)\rangle, n) \lhd old \neq new \rhd \delta + \\ & \quad \textstyle\sum_{d:D} r(d)\cdot q(old, new+1, v + \langle new, d\rangle, n) \lhd old + n \neq new \rhd \delta\end{aligned}$$

$$\begin{aligned}\textbf{proc}\quad & q'(l{:}list, n{:}\mathbb{N}) = \\ & \quad s(toe(l))\cdot q'(untoe(l), n) \lhd size(l) > 0 \rhd \delta + \\ & \quad \textstyle\sum_{d:D} r(d)\cdot q'(cons(d, l), n) \lhd size(l) < n \rhd \delta\end{aligned}$$

In the definition of the process q, *set* is a data type defining sets, where $\emptyset$ is the empty set and $+$ and $-$ denote addition and deletion, respectively. Elements of the set v are pairs $\langle i, d\rangle$ consisting of an index i and a datum d, where the index uniquely determines the datum (intuition: the oldest datum has the smallest index). Moreover, $get(v, i)$ is the unique datum d such that $\langle i, d\rangle \in v$ (if v contains exactly one $\langle i, d\rangle$, otherwise $get(v, i)$ is arbitrary). In the definition of process q', *list* is the common data type of lists, with constructors *nil* and *cons*. For brevity, we denote lists with square brackets: [] for the empty list and $[d, d_0, \ldots, d_{n-1}]$ for $cons(d, [d_0, \ldots, d_{n-1}])$. Moreover, $toe(l)$ is the last element of the list l and $untoe(l)$ is the result of deleting this element ($toe([])$ and $untoe([])$ are arbitrary). Finally, $size(l)$ is the length of the list l.

We want to prove $q(0, 0, \emptyset, n) = q'([], n)$ using the Concrete Invariant Corollary 3.9. We use the following CLPO Φ

$$\begin{aligned}\Phi = \lambda & old, new{:}\mathbb{N}\, v{:}set\, n{:}\mathbb{N}. \\ & s(get(v, old))\cdot q(old+1, new, v - \langle old, get(v, old)\rangle, n) \lhd old \neq new \rhd \delta + \\ & \textstyle\sum_{d:D} r(d)\cdot q(old, new+1, v + \langle new, d\rangle, n) \lhd old + n \neq new \rhd \delta\end{aligned}$$

which is exactly the CLPO defining q. Now we define the relation $I(old, new, v, n)$ by

$$old \leq new \leq old + n \ \wedge v = \{\langle old, get(v, old)\rangle, \ldots, \langle new - 1, get(v, new - 1)\rangle\}.$$

This relation expresses the intuition about the correct functioning of the first queue: there can never be more than n elements in the list, i.e. *old* and *new* do not differ more than n, and all positions from *old* to *new* contain a datum. This relation is an invariant in the sense of Corollary 3.9 as it satisfies

(a1) $(I(x, y, v, n) \wedge x \neq y) \rightarrow I(x+1, y, v - \langle x, get(v, x)\rangle, n)$;

(a2) $(I(x, y, v, n) \wedge x + n \neq y) \rightarrow I(x, y+1, v + \langle y, d\rangle, n)$;

Next we show that q and $q'' = \lambda old, new{:}\mathbb{N}\, v{:}set\, n{:}\mathbb{N}.q'(\alpha(old, new, v), n)$ satisfy

$$I(x,y,v,n) \rightarrow q(x,y,v,n) = (\Phi q)(x,y,v,n),$$
$$I(x,y,v,n) \rightarrow q''(x,y,v,n) = (\Phi q'')(x,y,v,n).$$

If $new \geq old$, $\alpha(old, new, v)$ denotes $[get(v, new-1), \ldots, get(v, old)]$. If $new < old$, we let $\alpha(old, new, v)$ denote $[]$. The parameterised process q is by definition a fixed point of Φ. We do not even need the invariant to show this. For the other implication we calculate

$$\begin{aligned} q''(old, new, v, n) &= q'(\alpha(old, new, v), n) = \\ &\quad s(toe(\alpha(old, new, v)))\cdot q'(untoe(\alpha(old, new, v), n) \triangleleft size(\alpha(old, new, v)) > 0 \triangleright \delta + \\ &\quad \Sigma_{d:D} r(d)\cdot q'(cons(d, \alpha(old, new, v)), n) \triangleleft size(\alpha(old, new, v)) < n \triangleright \delta. \end{aligned}$$

Now observe that the invariant $I(x,y,v,n)$ implies the following identities.

(b1) $(x \neq y) = (size(\alpha(x,y,v)) > 0)$;

(b2) $(x + n \neq y) = (size(\alpha(x,y,v)) < n)$;

(c1) $(x \neq y) \rightarrow get(v,x) = toe(\alpha(x,y,v))$;

(c2) $(x + n \neq y) \rightarrow d = d$;

(d1) $(x \neq y) \rightarrow \alpha(x+1, y, v - \langle x, get(v,x)\rangle) = untoe(\alpha(x,y,v))$;

(d2) $(x + n \neq y) \rightarrow \alpha(x, y+1, v + \langle y, d\rangle) = cons(d, \alpha(x,y,v))$.

As an example we prove d2:

$$(I(x,y,v,n) \wedge x + n \neq y) \rightarrow \alpha(x, y+1, v + \langle y, d\rangle) =^{\star} [d, get(v, y-1), \ldots, get(v,x)] =$$

$$cons(d, [get(v, y-1), \ldots, get(v,x)]) = cons(d, \alpha(x,y,v)).$$

In $\star$ we use the invariant to show that $get(v + \langle y, d\rangle, y) = d$. From the identities above it easily follows that

$$\begin{aligned} &I(old, new, v, n) \rightarrow q'(\alpha(old, new, v), n) = \\ &\quad s(get(v, old))\cdot q'(\alpha(old+1, new, v - \langle old, get(v, old)\rangle), n) \triangleleft old \neq new \triangleright \delta + \\ &\quad \Sigma_{d:D} r(d)\cdot q'(\alpha(old, new+1, v + \langle new, d\rangle), n) \triangleleft old + n \neq new \triangleright \delta = \\ &(\Phi q'')(old, new, v, n). \end{aligned}$$

Now it follows from the Concrete Invariant Corollary 3.9 that

$$I(old, new, v, n) \rightarrow q(old, new, v, n) = q'(\alpha(old, new, v), n).$$

Because $I(0, 0, \emptyset, n)$ is satisfied we find

$$q(0, 0, \emptyset, n) = q'([], n).$$

Remark 4.2. As Corollary 3.9 follows from CL-RSP, the proof can be done without using it. This would require calculating with an explicit Ξ as used in Corollary 3.8. There are two reasons why this is unpleasant. In the first place it leads to larger terms that must be manipulated. In the second place it mixes checking the invariant properties with proving that processes are fixed points. The use of Corollary 3.9 yields a better separation of these two concerns.

SUM6	$\Sigma_{d:D}(p(d) \between z) = (\Sigma p) \between z$	CF	$a(d) \mid b(e) = \begin{cases} \gamma(a,b)(d) & \text{if } d = e \text{ and} \\ & \gamma(a,b) \text{ defined} \\ \delta & \text{otherwise} \end{cases}$	
SUM7	$\Sigma_{d:D}(p(d) \mid z) = (\Sigma p) \mid z$			
SUM8	$\Sigma_{d:D}(\partial_H(p(d))) = \partial_H(\Sigma p)$			
SUM9	$\Sigma_{d:D}(\tau_I(p(d))) = \tau_I(\Sigma p)$			
SUM10	$\Sigma_{d:D}(\rho_R(p(d))) = \rho_R(\Sigma p)$	CD1	$\delta \mid x = \delta$	
		CD2	$x \mid \delta = \delta$	
CM1	$x \parallel y = x \between y + y \between x + x \mid y$	CT1	$\tau \mid x = \delta$	
CM2	$c \between x = c{\cdot}x$	CT2	$x \mid \tau = \delta$	
CM3	$c{\cdot}x \between y = c{\cdot}(x \parallel y)$			
CM4	$(x + y) \between z = x \between z + y \between z$	DD	$\partial_H(\delta) = \delta$	
CM5	$c{\cdot}x \mid c' = (c \mid c'){\cdot}x$	DT	$\partial_H(\tau) = \tau$	
CM6	$c \mid c'{\cdot}x = (c \mid c'){\cdot}x$	D1	$\partial_H(a(d)) = a(d)$	if $a \notin H$
CM7	$c{\cdot}x \mid c'{\cdot}y = (c \mid c'){\cdot}(x \parallel y)$	D2	$\partial_H(a(d)) = \delta$	if $a \in H$
CM8	$(x + y) \mid z = x \mid z + y \mid z$	D3	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	
CM9	$x \mid (y + z) = x \mid y + x \mid z$	D4	$\partial_H(x{\cdot}y) = \partial_H(x){\cdot}\partial_H(y)$	

Table 2: $x, y, z \in \mathbb{P}$, $a, b \in pAct$, $d, e \in D$ for data type D and c, c' are $a(d)$, τ or δ

4.2 μCRL^1-algebras

The example in the next subsection shows how invariants can be combined straightforwardly with parallelism. In this section we introduce the required operators. Besides the set $pAct$ we now also assume a partial function $\gamma : pAct \to pAct \to pAct$ that defines how actions communicate. The function γ is commutative and associative and if $\gamma(a,b) = c$, then the data types of a, b and c are all equal.

Definition 4.3. A μCRL^1-*algebra* is a set $\mathbb{P}$ with the operations

$a : D \to \mathbb{P}$ (for all $a \in pAct$ and data type D)
$\delta, \tau : \mathbb{P}$
$+, \cdot : \mathbb{P} \to \mathbb{P} \to \mathbb{P}$
$\Sigma : (D \to \mathbb{P}) \to \mathbb{P}$ (for each data type D)
$\parallel, \between, \mid : \mathbb{P} \to \mathbb{P} \to \mathbb{P}$
$\_ \lhd \_ \rhd \_ : (\mathbb{P} \times \mathbf{Bool} \times \mathbb{P}) \to \mathbb{P}$
$\partial_H, \tau_I, \rho_R : \mathbb{P} \to \mathbb{P}$ (for all $H, I \subseteq pAct$ and $R \in pAct \to pAct$ such that if $R(a) = b$ and $a : D \to \mathbb{P}$ then $b : D \to \mathbb{P}$)

that satisfies the (conditional) equations in Tables 1 through 3, except B1 and B2. If B1 and B2 are also satisfied, then we call $\mathbb{P}$ a μCRL$^{1\tau}$-*algebra*.

The name of a μCRL^1-algebra is derived from the specification language μCRL [7], because the algebra contains the same process operators. The superscript 1 refers to the fact that actions can have only one parameter. For an explanation of the new operators we refer to [2].

TID	$\tau_I(\delta) = \delta$		RD	$\rho_R(\delta) = \delta$
TIT	$\tau_I(\tau) = \tau$		RT	$\rho_R(\tau) = \tau$
TI1	$\tau_I(a(d)) = a(d)$	if $a \notin I$	R1	$\rho_R(a(d)) = R(a)(d)$
TI2	$\tau_I(a(d)) = \tau$	if $a \in I$		
TI3	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$		R3	$\rho_R(x + y) = \rho_R(x) + \rho_R(y)$
TI4	$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$		R4	$\rho_R(x \cdot y) = \rho_R(x) \cdot \rho_R(y)$

Table 3: $x, y \in \mathbb{P}$, $a \in pAct$ and $d \in D$ for some data type D

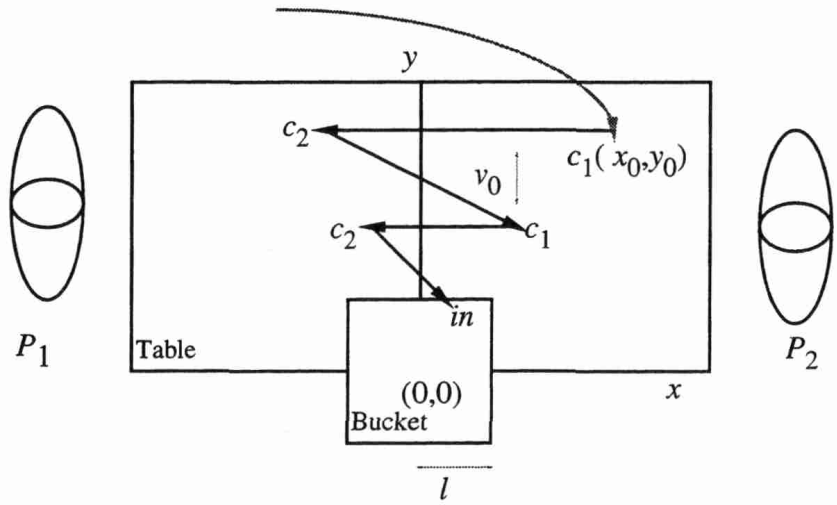


Figure 1: Players p_1 and p_2 play table tennis

A process operator Ψ is called (μCRL^1-)*expressible* iff Ψ can be expressed with the operations mentioned in Definition 4.3. A μCRL^1-algebra is said to satisfy RDP iff every expressible process operator Ψ has at least one fixed point.

4.3 Table tennis

Example 4.4. We consider two persons playing table tennis. Person p_1 is a perfect player who always returns the ball properly. Person p_2 is a weak player who misses the ball when it is too far. The goal of the game is to get the ball into a square bucket that stands on the table.

When we assume that the players play rather rigidly, we can describe the behaviour of the players as follows:

$$\begin{aligned}\textbf{proc}\quad & p_1(l, v{:}\mathbb{R}) = \textstyle\sum_{x,y:\mathbb{R}} r_2(x,y)\,(in \lhd |x| < l \wedge |y| < l \rhd s_1(-x, y - v)\,p_1(l, v)) \\ & p_2(x_h, y_h, k, l{:}\mathbb{R}) = \textstyle\sum_{x,y:\mathbb{R}} r_1(x,y)\,(in \lhd |x| < l \wedge |y| < l \rhd \\ & \qquad (s_2(-\tfrac{1}{2}x, y)\,p_2(x, y, k, l) \lhd (x_h - x)^2 + (x_h - y)^2 < k^2 \rhd out))\end{aligned}$$

A referee throws the ball at position x_0, y_0. Player p_2 receives the ball (via the communication $\gamma(r_1, s_1) = c_1$) and sends it back to position $(-\frac{1}{2}x_0, y_0)$. Then player p_1 receives the ball (via communication $\gamma(r_2, s_2) = c_2$ and sends it back to position $(\frac{1}{2}x_0, y_0 - v_0)$. The game continues, except if the ball arrives in the square bucket, which is placed around position $(0, 0)$ with sides $2l$, or if player p_2 cannot move his hand far enough from position (x_h, y_h) to the ball, which he can only do over a distance k each time it is his turn.

The total system of players is described by the following expression, where all communication actions are hidden, because we are only interested in the question whether the ball will end up on the floor or in the bucket.

$$Syst(x_0, y_0, v_0, x_h, y_h, k, l) = \tau_{\{c_1,c_2\}}(\partial_{\{r_1,r_2,s_1,s_2\}}(s_1(x_0, y_0)\cdot p_1(l, v_0) \parallel p_2(x_h, y_h, k, l))).$$

We sketch the main steps towards an answer of the question above. Using the axioms above we first expand *Syst* to the following equation, referred to as (I).

$$\begin{aligned}
&Syst(x_0, y_0, v_0, x_h, y_h, k, l) = \\
&\quad \tau\cdot in\cdot\delta \lhd (|x_0| < l \wedge |y_0| < l) \vee \\
&\qquad\qquad ((x_h - x_0)^2 + (y_h - y_0)^2 < k^2 \wedge |\tfrac{1}{2}x_0| < l \wedge |y_0| < l) \rhd \delta + \\
&\quad \tau\cdot out\cdot\delta \lhd (|x_0| \geq l \vee |y_0| \geq l) \wedge (x_h - x_0)^2 + (y_h - y_0)^2 \geq k^2 \rhd \delta + \\
&\quad \tau\cdot Syst(\tfrac{1}{2}x_0, y_0 - v_0, v_0, x_0, y_0, k, l) \\
&\qquad\qquad \lhd (|\tfrac{1}{2}x_0| \geq l \vee |y_0| \geq l) \wedge (x_h - x_0)^2 + (y_h - y_0)^2 < k^2 \rhd \delta.
\end{aligned}$$

Note that the definition of *Syst* cannot be seen as a guarded recursive specification in the usual sense, due to the unguarded occurrence of *Syst* on the right hand side. We develop sufficient conditions for a successful game. In order to guarantee that player p_2's hand is close enough to the place where the ball will land, and that the speed of the ball is within reasonable limits (otherwise the ball will end too far away in the y-direction), we formulate the following condition (II)

$$(x_h - x_0)^2 + (x_h - y_0)^2 < k^2 \wedge \tfrac{1}{4}x_0^2 + v_0^2 < k^2.$$

Moreover, the ball must end in the bucket. This is expressed by (III)

$$n \geq 0 \wedge 2^n l > x_0 \geq 0, \text{ where } n = \text{entier}(\frac{y_0 + l}{v_0}).$$

The variable n expresses how many turns it takes to get the ball in the bucket. It is easy to see that the conjunction of (II) and (III) is an invariant in the sense of Corollary 3.9.

Moreover, starting in a state where the invariant holds, a game only takes n turns. It follows that the convergence condition of Corollary 3.9 is satisfied.

The parameterised process $\lambda x_0, y_0, v_0, x_h, y_h, k, l{:}\mathbb{R}.\tau\cdot in\cdot\delta$ is a solution of (I) if (II) and (III) hold. Moreover, *Syst* is a solution for (I) by definition. Hence it follows from Corollary 3.9 that

$$(\text{II}) \wedge (\text{III}) \rightarrow Syst(x_0, y_0, v_0, x_h, y_h, k, l) = \tau\cdot in\cdot\delta.$$

References

[1] J.C.M. Baeten, editor. *Applications of Process Algebra.* Cambridge Tracts in Theoretical Computer Science 17. Cambridge University Press, 1990.

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

[3] M.A. Bezem and J.F. Groote. A correctness proof of a one bit sliding window protocol in μCRL. Technical report, Logic Group Preprint Series, Utrecht University, 1993. To appear.

[4] K.M. Chandy and J. Misra. *Parallel Program Design. A Foundation.* Addison-Wesley, 1988.

[5] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In G.X. Ritter, editor, *Information Processing 89*, pages 613–618. North-Holland, 1989. Full version available as Report CS-R9120, CWI, Amsterdam, 1991.

[6] R.A. Groenveld. Verification of a sliding window protocol by means of process algebra. Report P8701, Programming Research Group, University of Amsterdam, 1987.

[7] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. Technical Report CS-R9076, CWI, Amsterdam, December 1990.

[8] J.F. Groote and A. Ponse. Process algebra with guards. Combining Hoare logic and process algebra (Extended abstract). In J.C.M. Baeten and J.F. Groote, editors, *Proceedings CONCUR 91,* Amsterdam, volume 527 of *Lecture Notes in Computer Science*, pages 235–249. Springer-Verlag, 1991. Full version appeared as Technical Report CS-R9069, CWI, Amsterdam, 1990; and is to appear in *Formal Aspects of Computing.*

[9] J.F. Groote and A. Ponse. Proof theory for μCRL. Technical Report CS-R9138, CWI, Amsterdam, August 1991.

[10] E.C.R. Hehner. **do** Considered **od**: A contribution to the programming calculus. *Acta Informatica*, 11:287–304, 1979.

[11] A. Ponse. Process expressions and Hoare's logic. *Information and Computation*, 95(2):192–217, 1991.