**University of Amsterdam**

*Programming Research Group*

# A Constructive Version of AIP Revisited

A. Barros
T. Hou

Report PRG0802 January 2008

A. Barros

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

tel. +31 20 525.7585
e-mail: barros@science.uva.nl

T. Hou

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

tel. +31 20 525.7585
e-mail: hou@science.uva.nl

Programming Research Group Electronic Report Series

# A Constructive Version of AIP Revisited

Alexandra Barros and Tie Hou

Informatics Institute, University of Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{barros, hou}@science.uva.nl

**Abstract**

In this paper, we review a constructive version of the Approximation Induction Principle. This version states that bisimilarity of regular processes can be decided by observing only a part of their behaviour. We use this constructive version to formulate a complete inference system for the Algebra of Communicating Processes with linear recursive specifications.

## 1 Introduction

The *Approximation Induction Principle* (AIP) was introduced by Baeten, Bergstra and Klop [2]. This proof rule states that if all finite projections of two processes are equal, then the two processes are equal. The projection of a process at level $n$ can execute all transitions of this process up to the first $n$ steps. At first sight, AIP can be used to test the equality of processes. However, as there can be infinitely many finite projections of a process, it is not possible to derive the equality of processes by testing their projections one by one.

In 1987, Mauw reformulated AIP into a constructive version of this principle (AIP$^c$), proving its correctness for regular processes [7]. He pointed out that not all projections need to be considered for deriving the equality of regular processes, but only a certain projection at a large enough level is sufficient. Although not well known, AIP$^c$ leads to a valuable result: it can be used to formulate an effective inference system for regular processes and to consider algorithms to decide upon equality.

In this paper, we review AIP$^c$ in the context of the Algebra of Communicating Processes. We give a more general view of this principle by considering the deadlock constant $\delta$, introduced by Bergstra and Klop [4]. Mauw's work does not consider this constant or the notions of parallelism and concurrency. We also use the notation for a solution of a linear recursive specification introduced by van Glabbeek [10] in our definition and proof of AIP$^c$.

The paper is organized as follows. In Section 2, we describe some basic topics in process algebra. Next, in Section 3, we discuss the revisited constructive version of AIP and give the bound for the number of transition steps to be considered. In Section 4, we prove that, in order to derive the equality of processes, it is not possible to reduce this bound. We present a complete proof system for ACP (Algebra of Communicating Processes) [4] with linear recursive specifications in Section 5. A comparison of Mauw's approach and ours, and some concluding remarks are described in Section 6.

# 2 Process Algebra

ACP is an axiom system. Models for ACP can be constructed using several techniques, such as process graphs, the term model, and the projective limit model. A description of those models can be found in the work of Baeten and Weijland [3]. We consider processes as elements of a model for ACP.

BPA (Basic Process Algebra) is a widely known subsystem of ACP which does not consider deadlock, encapsulation and the operators for concurrency. We begin by describing an extension of BPA. Next, we present the ACP axiom system.

## 2.1 Basic Process Algebra with deadlock

BPA$_\delta$ extends BPA with the distinction between successful and unsuccessful termination. Consider a process that has reached a state in which it has stopped executing actions and cannot proceed. This state is referred to as *deadlock*.

Consider a non-empty set of *actions* A with $\delta \notin$ A. We denote A $\cup \{\delta\}$ by A$_\delta$. The signature of BPA$_\delta$ consists of the following operators:

1. for every $a \in$ A, an *action* constant $a$;

2. the *deadlock* constant $\delta$;

3. *variables* $x, y, z, \ldots$;

4. the binary *alternative composition* operator $+$;

5. the binary *sequential composition* operator $\cdot$.

The axioms of BPA$_\delta$ are listed in Table 1.

Table 1: Axioms of BPA$_\delta$

| | | | |
|---|---|---|---|
| A1 | $x + y$ | $=$ | $y + x$ |
| A2 | $(x + y) + z$ | $=$ | $x + (y + z)$ |
| A3 | $x + x$ | $=$ | $x$ |
| A4 | $(x + y) \cdot z$ | $=$ | $x \cdot z + y \cdot z$ |
| A5 | $(x \cdot y) \cdot z$ | $=$ | $x \cdot (y \cdot z)$ |
| A6 | $x + \delta$ | $=$ | $x$ |
| A7 | $\delta \cdot x$ | $=$ | $\delta$ |

As binding convention, the operator $\cdot$ binds stronger than the operator $+$. We can omit the operator $\cdot$ from terms, i.e, instead of writing $a \cdot b$ we can simply write $ab$. Let $s$ and $t$ range over terms. The symbols in the signature can be informally explained as follows:

1. $a$ executes action $a$ and then terminates successfully;
2. $\delta$ does not display any behaviour and terminates unsuccessfully.
3. $s + t$ executes either $s$ or $t$;
4. $s \cdot t$ first executes $s$, and upon successful termination of $s$ executes $t$.

We formalize this intuition by applying *structural operational semantics* as described in the work of Aceto, Fokkink and Verhoef [1]. This includes giving a collection of transition rules which defines transitions $t \xrightarrow{a} t'$ to express that term $t$ can evolve into term $t'$ by the execution of action $a$, and predicates $t \xrightarrow{a} \surd$ to express that term $t$ can terminate successfully by the execution of action $a$. The transition rules for BPA$_\delta$ are presented in Table 2. The variables $x$, $x'$, $y$, and $y'$ range over closed terms and $a$ ranges over actions in $A$. Since deadlock does not display any behaviour, there is no transition rule for this constant.

Table 2: Transition rules for BPA$_\delta$ ($a \in$ A)

$$\frac{}{a \xrightarrow{a} \surd}$$

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

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

## 2.2 Algebra of Communicating Processes

The signature of ACP consists of the signature of BPA$_\delta$ plus the following operators:

1. the binary *merge* or *parallel composition* operator $\|$;
2. the binary *left merge* operator $\mathbin{\|\!\!\_}$;
3. the binary *communication merge* operator $|$;
4. for each $H \subseteq$ A, a unary *encapsulation* operator $\partial_H$.

The operators $\parallel$, $\mathbin{\parallel\!\!\!\lfloor}$, and $\mid$ bind weaker than the $\cdot$ and stronger than the $+$.

The axioms of ACP consist of the axioms of $\mathrm{BPA}_\delta$ plus the axioms listed in Table 3, where $a, b \in \mathrm{A}_\delta$ and $H \subseteq \mathrm{A}$. We consider a *communication function* $\gamma$, which is a partial binary function on A that produces for certain pairs of actions $a$ and $b$ their communication $\gamma(a, b)$. The function $\gamma$ satisfies the following conditions:

1. for every $a, b \in \mathrm{A}$, if $\gamma(a, b)$ is defined, then $\gamma(a, b) = \gamma(b, a)$, i.e., $\gamma$ is commutative;

2. for every $a, b, c \in \mathrm{A}$, if $\gamma(\gamma(a, b), c)$ is defined, then $\gamma(\gamma(a, b), c) = \gamma(a, \gamma(b, c)$, i.e., $\gamma$ is associative.

Note that condition 2 implies that if for certain actions $a$, $b$, and $c$, $\gamma(a, b)$ and $\gamma(\gamma(a, b), c)$ are both defined, then also $\gamma(b, c)$ and $\gamma(a, \gamma(b, c))$ are defined.

Table 3: Axioms for merge with communication and encapsulation

| | | | |
|---|---|---|---|
| CF1 | $a \mid b$ | $=$ | $\gamma(a, b)$, if $\gamma(a, b)$ is defined; |
| CF2 | $a \mid b$ | $=$ | $\delta$ otherwise. |
| CM1 | $x \parallel y$ | $=$ | $(x \mathbin{\parallel\!\!\!\lfloor} y + y \mathbin{\parallel\!\!\!\lfloor} x) + x \mid y$ |
| CM2 | $a \mathbin{\parallel\!\!\!\lfloor} x$ | $=$ | $a \cdot x$ |
| CM3 | $a \cdot x \mathbin{\parallel\!\!\!\lfloor} y$ | $=$ | $a \cdot (x \parallel y)$ |
| CM4 | $(x + y) \mathbin{\parallel\!\!\!\lfloor} z$ | $=$ | $x \mathbin{\parallel\!\!\!\lfloor} z + y \mathbin{\parallel\!\!\!\lfloor} z$ |
| CM5 | $(a \cdot x) \mid b$ | $=$ | $(a \mid b) \cdot x$ |
| CM6 | $a \mid (b \cdot x)$ | $=$ | $(a \mid b) \cdot x$ |
| CM7 | $(a \cdot x) \mid (b \cdot y)$ | $=$ | $(a \mid b) \cdot (x \parallel y)$ |
| CM8 | $(x + y) \mid z$ | $=$ | $x \mid z + y \mid z$ |
| CM9 | $x \mid (y + z)$ | $=$ | $x \mid z + y \mid z$ |
| D1 | $\partial_H(a)$ | $=$ | $a$, if $a \notin H$ |
| D2 | $\partial_H(a)$ | $=$ | $\delta$, if $a \in H$ |
| D3 | $\partial_H(x + y)$ | $=$ | $\partial_H(x) + \partial_H(y)$ |
| D4 | $\partial_H(x \cdot y)$ | $=$ | $\partial_H(x) \cdot \partial_H(y)$ |

The symbols in the signature can be informally explained as follows:

1. $s \parallel t$ executes both $x$ and $y$ in parallel;

2. $s \mathbin{\parallel\!\!\!\lfloor} t$ executes an initial transition from $s$, and then executes the merge of the remaining part of $s$ and $t$;

3. $s \mid t$ executes as initial transition a communication between initial transitions of $s$ and $t$, and then executes the merge of the remaining parts of $s$ and $t$.

4. $\partial_H$ renames all actions in $H$ into $\delta$.

The transition rules for the merge with communication and encapsulation are presented in Table 4.

Table 4: Transition rules for merge with communication and encapsulation, where $\gamma(a, b)$ is defined.

$$\frac{x \xrightarrow{a} \surd}{x \| y \xrightarrow{a} y} \qquad \frac{x \xrightarrow{a} x'}{x \| y \xrightarrow{a} x' \| y} \qquad \frac{y \xrightarrow{a} \surd}{x \| y \xrightarrow{a} x} \qquad \frac{y \xrightarrow{a} y'}{x \| y \xrightarrow{a} x \| y'}$$

$$\frac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} \surd}{x \| y \xrightarrow{\gamma(a,b)} \surd} \qquad \frac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} y'}{x \| y \xrightarrow{\gamma(a,b)} y'} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} \surd}{x \| y \xrightarrow{\gamma(a,b)} x'} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x \| y \xrightarrow{\gamma(a,b)} x' \| y'}$$

$$\frac{x \xrightarrow{a} \surd}{x \lfloor\!\lfloor y \xrightarrow{a} y} \qquad \frac{x \xrightarrow{a} x'}{x \lfloor\!\lfloor y \xrightarrow{a} x' \| y}$$

$$\frac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} \surd}{x | y \xrightarrow{\gamma(a,b)} \surd} \qquad \frac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} y'}{x | y \xrightarrow{\gamma(a,b)} y'} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} \surd}{x | y \xrightarrow{\gamma(a,b)} x'} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x | y \xrightarrow{\gamma(a,b)} x' \| y'}$$

$$\frac{x \xrightarrow{a} \surd}{\partial_H(x) \xrightarrow{a} \surd} \; a \notin H \qquad \frac{x \xrightarrow{a} x'}{\partial_H(x) \xrightarrow{a} \partial_H(x')} \; a \notin H$$

## 2.3 Linear Recursive Specifications

A (*finite*) *recursive specification* $E$ is a set of recursive equations $\{X_i = t_i(\vec{X}) \mid X_i \in V_E\}$ where, for some $n \geq 1$, $V_E = \{X_1, X_2, \ldots, X_n\}$ is a set of *recursion variables*, $\vec{X}$ is a vector containing all recursion variables in $V_E$, i.e, $\vec{X} = X_1, \ldots, X_n$, and $t_i$ is a term over the signature of ACP which may contain the variables in $\vec{X}$. Infinite processes can be defined using recursive equations, for example, the equation $X = a \cdot X + b$ defines the process that executes action $a$ an arbitrary number of times. Each time it has the choice of executing $b$, after which it terminates.

A *solution* for a recursive equation is a process that solves the equation. We use the constant $\langle X_i | E \rangle$ to denote, in the term model and in the transition rules, the solution for the recursive equation $(X_i = t_i(\vec{X})) \in E$. Once a recursive specification $E$ is declared, $\langle X_i | E \rangle$ can be abbreviated by $\langle X_i \rangle$. We also talk about a solution for a recursive specification $E$. A solution for $E$, with $V_E = \{X_1, \ldots, X_n\}$, is a vector $\langle X_1 | E \rangle, \ldots, \langle X_n | E \rangle$ such that substituting each variable in $V_E$ by its respective solution turns all equations in $E$ into true statements.

**Definition 1** (Linear Recursive Specification)**.** *A recursive specification* $E = \{X_i = t_i(\vec{X}) | i = 1, \ldots, n\}$ *is called linear if all its equations are linear, i.e., of the form*

$$X_i = \sum_{j=1}^{n} \alpha_{i,j} X_j + \beta_i,$$

*where $\alpha_{i,j}$ and $\beta_i$ are finite sums of actions or $\delta$.*

In practice we often omit the $\delta$-summands.

The signature of $\mathrm{ACP}_{\mathrm{lin}}$, where lin stands for linear recursion, consists of the signature of ACP plus for all linear recursive specifications $E$ and for all $X_i \in V_E$ a constant $\langle X_i|E\rangle$. The axioms of $\mathrm{ACP}_{\mathrm{lin}}$ consist of the axioms of ACP plus the *recursive definition principle* (RDP), presented in Table 5, where the linear recursive specification $E$ is the set of equations $\{X_i = t_i(\vec{X}) \mid i = 1, \ldots, n\}$. RDP states that $\langle X_1|E\rangle, \ldots, \langle X_n|E\rangle$ is a solution for $E$.

Table 5: Axiom for linear recursive specifications

| | |
|---|---|
| RDP | $\langle X_i\|E\rangle \quad = \quad t_i(\langle X_1\|E\rangle, \ldots, \langle X_n\|E\rangle) \quad (i \in \{1, \ldots, n\})$ |

The transition rules for $\mathrm{ACP}_{\mathrm{lin}}$ are obtained by extending the set of transition rules for ACP, given in Table 2 and Table 4, with the rules in Table 6.

Table 6: Transition rules for $\mathrm{ACP}_{\mathrm{lin}}$

$$\frac{t_i(\langle X_1|E\rangle, \ldots, \langle X_n|E\rangle) \xrightarrow{a} \surd}{\langle X_i|E\rangle \xrightarrow{a} \surd} \qquad \frac{t_i(\langle X_1|E\rangle, \ldots, \langle X_n|E\rangle) \xrightarrow{a} y}{\langle X_i|E\rangle \xrightarrow{a} y}$$

## 2.4 Projection

For each natural number $n$, the unary *projection* operator $\pi_n$ will block all further actions after $n$ actions have been executed. The signature of $\mathrm{ACP}_{\mathrm{lin}}$ + PR, where PR stands for projection, consists of the signature of $\mathrm{ACP}_{\mathrm{lin}}$ plus for each natural number $n$ a unary function $\pi_n$. Table 7 presents the axioms for the projection operator, where $a \in \mathrm{A}_\delta$.

Table 7: Axioms for projection operators

| | | | |
|---|---|---|---|
| PR1 | $\pi_{n+1}(a)$ | $=$ | $a$ |
| PR2 | $\pi_0(x)$ | $=$ | $\delta$ |
| PR3 | $\pi_{n+1}(a \cdot x)$ | $=$ | $a \cdot \pi_n(x)$ |
| PR4 | $\pi_n(x + y)$ | $=$ | $\pi_n(x) + \pi_n(y)$ |

The transition rules for $\mathrm{ACP}_{\mathrm{lin}}$ + PR is obtained by extending the set of transition rules of $\mathrm{ACP}_{\mathrm{lin}}$, Table 2, Table 4 and Table 6, with the rules in Table 8.

The Approximation Induction Principle (AIP) states that if every finite projection of two processes is equal, then the two processes are equal:

$$\text{(AIP)} \qquad \text{If } \pi_n(x) = \pi_n(y) \text{ for all } n \in \mathbf{N}, \text{ then } x = y.$$

Table 8: Transition rules for $\mathrm{ACP_{lin}} + \mathrm{PR}$

$$\frac{x \xrightarrow{a} \surd}{\pi_{n+1}(x) \xrightarrow{a} \surd} \qquad \frac{x \xrightarrow{a} x'}{\pi_{n+1}(x) \xrightarrow{a} \pi_n(x')}$$

At first sight, this principle could be used to derive the equality of processes. However, as processes have infinitely many projections, it is not possible to derive their equality by testing the projections one by one.

A corollary to AIP is that $t \parallel s = s \parallel t$, i.e., the merge is commutative for recursive terms.

# 3 A Constructive Version of AIP

A constructive version of AIP ($\mathrm{AIP^c}$) can be derived for the class of regular processes. $\mathrm{AIP^c}$ states that only a finite number of transition steps need to be considered in order to derive the equality of two regular processes.

Note that the solution for a linear equation in a linear recursive specification is a regular process. Consider a single linear recursive specification and two regular processes, each a solution for a linear equation. This way, the number of transition steps to be considered is equal to the number of linear equations in the linear recursive specification minus one.

**Theorem 1** ($\mathrm{AIP^c}$)**.** *Let $E$ be a linear recursive specification with $n$ variables and let $X_p$ and $X_q$ be two recursion variables in $V_E$, then:*

$$\pi_{n-1}(\langle X_p \rangle) = \pi_{n-1}(\langle X_q \rangle) \quad \Rightarrow \quad \langle X_p \rangle = \langle X_q \rangle$$

*Proof.* Consider the relation $\cong_k$ on $\langle V_E \rangle = \{\langle X \rangle | X \in V_E\}$ for $k \geq 0$ defined by:

$$\langle X_p \rangle \cong_k \langle X_q \rangle \quad \Leftrightarrow \quad \pi_k(\langle X_p \rangle) = \pi_k(\langle X_q \rangle).$$

Because

$$\pi_{k+1}(\langle X_p \rangle) = \pi_{k+1}(\langle X_q \rangle) \quad \Rightarrow \quad \pi_k(\langle X_p \rangle) = \pi_k(\langle X_q \rangle),$$

we can define the non-increasing sequence of relations:

$$\cong_0 \quad \supseteq \quad \cong_1 \quad \supseteq \quad \cong_2 \quad \ldots$$

The relation $\cong_0$ has only one equivalence class which is $\langle V_E \rangle$ itself. This is easily seen considering axiom PR2.

From now on, we need to prove two claims: (1) once this sequence becomes constant it remains constant; (2) the sequence is constant from $\cong_{n-1}$ at most.

**Claim 1.** $(\cong_k \quad = \quad \cong_{k+1}) \quad \Rightarrow \quad (\cong_{k+1} \quad = \quad \cong_{k+2})$.

*Proof.* Since $\cong_{k+1} \supseteq \cong_{k+2}$ by the relation's definition, we only need to prove that $\cong_{k+1} \subseteq \cong_{k+2}$. If we assume $\cong_{k+1} \not\subseteq \cong_{k+2}$, there are $X_p$ and $X_q$ in $V_E$ such that:

$$\pi_{k+1}(\langle X_p \rangle) = \pi_{k+1}(\langle X_q \rangle) \text{ and } \pi_{k+2}(\langle X_p \rangle) \neq \pi_{k+2}(\langle X_q \rangle).$$

Now let $X_p$ and $X_q$ be defined by the linear equations

$$\begin{aligned} X_p &= \sum_{i=1}^{n} \alpha_i X_i + \beta \\ X_q &= \sum_{j=1}^{n} \gamma_j X_j + \theta, \end{aligned}$$

where all $X_i$ and $X_j$ are elements of $V_E$. Then the projections at level $k+2$ are

$$\begin{aligned} \pi_{k+2}(\langle X_p \rangle) &= \sum_{i=1}^{n} \alpha_i \cdot \pi_{k+1}(\langle X_i \rangle) + \beta \\ \pi_{k+2}(\langle X_q \rangle) &= \sum_{j=1}^{n} \gamma_j \cdot \pi_{k+1}(\langle X_j \rangle) + \theta. \end{aligned}$$

Because $\beta$ is equal to $\theta$, it must be the case that:

$$\begin{array}{lll} \exists i \forall j & (\alpha_i \neq \gamma_j \vee \pi_{k+1}(\langle X_i \rangle) \neq \pi_{k+1}(\langle X_j \rangle)) & \vee \\ \exists j \forall i & (\alpha_i \neq \gamma_j \vee \pi_{k+1}(\langle X_i \rangle) \neq \pi_{k+1}(\langle X_j \rangle)). & \end{array}$$

Applying $\cong_k = \cong_{k+1}$ on the above expression:

$$\begin{array}{lll} \exists i \forall j & (\alpha_i \neq \gamma_j \vee \pi_{k}(\langle X_i \rangle) \neq \pi_{k}(\langle X_j \rangle)) & \vee \\ \exists j \forall i & (\alpha_i \neq \gamma_j \vee \pi_{k}(\langle X_i \rangle) \neq \pi_{k}(\langle X_j \rangle)). & \end{array}$$

So this expression must be true in order for $\pi_{k+2}(\langle X_p \rangle) \neq \pi_{k+2}(\langle X_q \rangle)$ to hold. From this we find:

$$\pi_{k+1}(\langle X_p \rangle) = \sum_{i=1}^{n} \alpha_i \cdot \pi_k(\langle X_i \rangle) + \beta \quad \neq$$
$$\sum_{j=1}^{n} \gamma_j \cdot \pi_k(\langle X_j \rangle) + \theta = \pi_{k+1}(\langle X_q \rangle).$$

This is in contradiction with the assumptions.

Hence,

$$(\cong_k = \cong_{k+1}) \quad \Rightarrow \quad (\cong_{k+1} = \cong_{k+2}).$$

□

**Claim 2.** $\cong_{n-1} = \cong_n$

*Proof.* Note that for two equivalence relations $R$ and $R'$, if $R \subset R'$, the number of equivalence classes generated by $R$ is strictly greater than the number of equivalence classes generated by $R'$. Now, since the sequence of relations defined by $\cong_k$ is initially strictly decreasing, and the maximum number of equivalence classes of $V_E$ is $n$ (one class for each term), we can conclude that at most the first $n$ relations in the sequence can be unequal, i.e., from $\cong_0$ to $\cong_{n-1}$. Therefore,

$$\cong_{n-1} \quad = \quad \cong_n .$$

□

Now we can use the two claims to infer from

$$\pi_{n-1}(\langle X_p \rangle) = \pi_{n-1}(\langle X_q \rangle)$$

the equality of all projections:

$$\forall k \geq 0 \quad \pi_k(\langle X_p \rangle) = \pi_k(\langle X_q \rangle).$$

By AIP we can conclude that $\langle X_p \rangle = \langle X_q \rangle$. □

# 4 Tightness

In this section, we show that the bound $n-1$ used in AIP$^{\mathrm{c}}$ for a linear recursive specification $E$ is *tight*, i.e., it is in general not possible to choose a lower value.

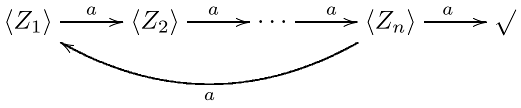
**Theorem 2.** *For any value $n \geq 2$, there is finite linear recursive specification $E$ with $n$ equations, and with $X_p, X_q \in V_E$, such that*

$$\pi_{n-2}(\langle X_p \rangle) = \pi_{n-2}(\langle X_q \rangle) \text{ and } \langle X_p \rangle \neq \langle X_q \rangle.$$

*Proof.* Take any $n \geq 2$. Construct the linear recursive specification $E$, with $V_E = \{Z_1, \ldots, Z_n\}$, as follows:

$$Z_k = \begin{cases} aZ_{k+1} & \text{if } k < n, \\ aZ_1 + a & \text{if } k = n. \end{cases}$$

The transition system associated to $E$, provided by the transition rules of ACP$_{\mathrm{lin}}$, is:



Now observe that

$$\pi_{n-2}(\langle Z_1 \rangle) = \pi_{n-2}(\langle Z_2 \rangle) = a^{n-2}\delta.$$

Although $\langle Z_1 \rangle$ and $\langle Z_2 \rangle$ are equal up to $n-2$ steps, $\langle Z_1 \rangle \neq \langle Z_2 \rangle$ because

$$\pi_{n-1}(\langle Z_1 \rangle) = a^{n-1}\delta \text{ while } \pi_{n-1}(\langle Z_2 \rangle) = a^{n-2}(a\delta + a).$$

So, on the basis of AIP$^{\mathrm{c}}$ we cannot conclude that $\langle Z_1 \rangle = \langle Z_2 \rangle$. □

As an example, consider $n = 4$. In this case, the linear recursive specification $E$ consists of the following equations:

$$\begin{array}{rcl} Z_1 & = & aZ_2 \\ Z_2 & = & aZ_3 \\ Z_3 & = & aZ_4 \\ Z_4 & = & aZ_1 + a \end{array}$$

The projections of $\langle Z_1 \rangle$ and $\langle Z_2 \rangle$ at level 2 are equal, while the projections at level 3, described next, are not:

$$\begin{array}{rcl} \pi_3(\langle Z_1 \rangle) & = & a^3\delta \\ \pi_3(\langle Z_2 \rangle) & = & a^2(a\delta + a) \end{array}$$

# 5 A Complete Proof System

Before giving the completeness of the axiom system $\mathrm{ACP_{lin}} + \mathrm{PR} + \mathrm{AIP^c}$, we first describe *bisimulation equivalence* [8, 5, 9].

**Definition 2** (Bisimulation)**.** *A* bisimulation relation $\mathfrak{B}$ *is a binary relation on processes such that:*

1. *if $p\mathfrak{B}q$ and $p \xrightarrow{a} p'$, then $q \xrightarrow{a} q'$ with $p'\mathfrak{B}q'$;*
2. *if $p\mathfrak{B}q$ and $q \xrightarrow{a} q'$, then $p \xrightarrow{a} p'$ with $p'\mathfrak{B}q'$;*
3. *if $p\mathfrak{B}q$ and $p \xrightarrow{a} \surd$, then $q \xrightarrow{a} \surd$;*
4. *if $p\mathfrak{B}q$ and $q \xrightarrow{a} \surd$, then $p \xrightarrow{a} \surd$.*

*Two processes $p$ and $q$ are* bisimilar*, denoted by $p \leftrightarrows q$, if there is a bisimulation relation $\mathfrak{B}$ such that $p\mathfrak{B}q$.*

It is a standard result that $\leftrightarrows$ is an *equivalence relation* and also a *congruence* with respect to the operators of ACP. Congruence is an essential property for bisimulation equivalence to enable giving an axiomatisation that is sound and complete modulo bisimulation equivalence.

From now on, we show the completeness of the axiom system $\mathrm{ACP_{lin}} + \mathrm{PR} + \mathrm{AIP^c}$. First, consider the following lemma.

**Lemma 1.** *For each process term $t$ in $\mathrm{ACP_{lin}} + \mathrm{PR} + \mathrm{AIP^c}$, there is a linear recursive specification $E$ such that $t = \langle X_1|E \rangle$ and $X_1 \in V_E$.*

*Proof.* We apply structural induction on term $t$. Let $t$, $t_1$ and $t_2$ be processes in $\mathrm{ACP_{lin}} + \mathrm{PR} + \mathrm{AIP^c}$.

1. If $t \in \mathrm{A}_\delta$, we take $E = \{X_1 = t\}$. So, by RDP, $t$ is derivably equal to $\langle X_1|E \rangle$.

2. If $t$ is of the form $\partial_H(t_1)$, by induction, there is a linear recursive specification $F$ defined by the following recursive equations, where $t_1$ is derivably equal to $\langle Y_1|F\rangle$:

$$Y_i = \sum_{j=1}^{n} \alpha_{ij} Y_j + \beta_i.$$

Then we define $E$ by the following recursive equations:

$$Y_i = \sum_{j=1}^{n} \gamma_{ij} Y_j + \theta_i,$$

where $V_E = V_F$ and $\gamma_{ij}$ and $\theta_i$ are derived from $\alpha_{ij}$ and $\beta_i$ by replacing every action that belongs to $H$ by $\delta$.

3. If $t$ is of the form $t_1 \Box t_2$, with $\Box \in \{+, \cdot, \parallel, \llfloor, \mid\}$, by induction, there are linear recursive specifications $F = \{Y_i = t_i(\vec{Y})|i = 1, \ldots, n\}$ and $G = \{Z_j = u_j(\vec{Z})|j = 1, \ldots, m\}$, such that $t_1$ and $t_2$ are derivably equal to $\langle Y_1|F\rangle$ and $\langle Z_1|G\rangle$, respectively. Without loss of generality, we assume that $V_F \cap V_G = \emptyset$.

First, consider the operator $+$. In this case, we take $E = \{X_1 = t_1(\vec{Y}) + u_1(\vec{Z})\} \cup F \cup G$, where $V_E = \{X_1\} \cup V_F \cup V_G$ and $X_1$ is not in $V_F \cap V_G$. So, by RDP, $t$ is derivably equal to $\langle X_1|E\rangle$ .

For the operator $\cdot$, we take $E = \{Y_i Z_1 = t_i(\vec{Y})Z_1|i = 1, \ldots, n\} \cup G$, where $V_E = V_G \cup \{Y_k Z_1|k = \{1, \ldots, n\}\}$. So, by RDP, $t$ is derivably equal to $\langle X_1|E\rangle$ .

For the $\parallel, \llfloor, \mid$, we need to use simultaneous induction and consider each operator separately. First, assume that $F$ and $G$ are defined by the following recursive equations:

$$\begin{aligned} Y_i &= \sum_{j=1}^{n} \alpha_{ij} Y_j + \beta_i \\ Z_l &= \sum_{k=1}^{m} \gamma_{lk} Z_k + \theta_l. \end{aligned}$$

We define for the merge operator, $\parallel$, the linear recursive specification $E$

by the following recursive equations:

$$\begin{aligned}
X_{pq} \quad = \quad & \sum_{j=1}^{n} \alpha_{pj} X_{jq} + \beta_p Z_q + \qquad (1)\\
& \sum_{k=1}^{m} \gamma_{qk} X_{pk} + \theta_q Y_p + \qquad (2)\\
& \sum_{j=1}^{n} \sum_{k=1}^{m} (\alpha_{pj} \mid \gamma_{qk}) X_{jk} + \qquad (3)\\
& \sum_{j=1}^{n} (\alpha_{pj} \mid \theta_q) Y_j + \qquad (4)\\
& \sum_{k=1}^{m} (\beta_p \mid \gamma_{qk}) Z_k + \qquad (5)\\
& (\beta_p \mid \theta_q), \qquad (6)
\end{aligned}$$

with $V_E = V_F \cup V_G \cup \{X_{pq} \mid 1 \leq p \leq n \text{ and } 1 \leq q \leq m\}$. We have that $t$ is derivably equal to $\langle X_{11} | E \rangle$. In this specification, lines 1 and 2 refer to the two left merge operators in axiom CM1 and lines 3 to 6 refer to the communication merge, more specifically: line 3 considers the case when a communication between a transition from $\alpha$ and a transition from $\gamma$ is done, line 4 when a communication between transitions from $\alpha$ and $\theta$ is done, line 5 between $\beta$ and $\gamma$, and line 6, between $\beta$ and $\theta$.

For the left merge $t1 \mathbin{\|\mkern-9mu\lfloor} t2$ we define a linear recursive specification $D$ that contains the equations in $E$ plus:

$$W = \sum_{j=1}^{n} \alpha_{1j} X_{j1} + \beta Z_1,$$

where $V_D = W \cup V_E$. This way, $t$ is derivably equal to $\langle W | D \rangle$. Note that

$$t_2 \mathbin{\|\mkern-9mu\lfloor} t_1 = \sum_{k=1}^{n} \gamma_{1k} (t_2 \parallel t_1) + \theta \cdot t_1.$$

However, by AIP, we may assume commutativity of $\|$.

For the communication operator we define a linear recursive specification

$D$ that contains the equations in $E$ plus:

$$\begin{aligned} W &= \sum_{j=1}^{n}\sum_{k=1}^{m}(\alpha_{1j} \mid \gamma_{1k})X_{jk} + \\ &\quad \sum_{j=1}^{n}(\alpha_{1j} \mid \theta_1)Y_j + \\ &\quad \sum_{k=1}^{m}(\beta_1 \mid \gamma_{1k})Z_k + \\ &\quad (\beta_1 \mid \theta_1), \end{aligned}$$

where $V_D = W \cup V_E$. This way, $t$ is derivably equal to $\langle W|D\rangle$.

□

**Theorem 3.** *The axiom system* $\mathrm{ACP_{lin}} + \mathrm{PR} + \mathrm{AIP^c}$ *is a sound and complete axiomatisation for the set of closed terms in* $\mathrm{ACP_{lin}} + \mathrm{PR} + \mathrm{AIP^c}$ *modulo bisimulation equivalence. That is, for closed terms $s$ and $t$ in* $\mathrm{ACP_{lin}} + \mathrm{PR} + \mathrm{AIP^c}$*:*

$$\mathrm{ACP_{lin}} + \mathrm{PR} + \mathrm{AIP^c} \vdash s = t \Leftrightarrow s \leftrightarroweq t.$$

*Proof.* $\Rightarrow$(Soundness):

We need to verify the soundness of each separate axiom. As shown in the work of Fokkink [6], soundness of A1 − 7, CF1 − 2, CM1 − 9, D1 − 4, RDP, PR1 − 4 and AIP is easy to prove. Soundness of $\mathrm{AIP^c}$ follows directly from Theorem 1 and from the soundness of AIP .

$\Leftarrow$(Completeness):

Let $s \leftrightarroweq t$. By Lemma 1, there is a recursive specification $E'$ with $X_1 \in V_{E'}$ and $s = \langle X_1|E'\rangle$ and a recursive specification $E''$ with $Y_1 \in V_{E''}$ and $t = \langle Y_1|E''\rangle$. Without loss of generality, we assume that $V_{E'} \cap V_{E''} = \emptyset$ and consider a single recursive specification $E$ such that $V_E = V_{E'} \cup V_{E''}$

Now we take $n = |V_E| - 1$. By definition, $\pi_n(s)$ and $\pi_n(t)$ can be equated to processes $s'$ and $t'$, respectively, in ACP. Since bisimulation equivalence is a congruence with respect to the operators of ACP, $s \leftrightarroweq t$ implies $\pi_n(s) \leftrightarroweq \pi_n(t)$. Soundness of the axioms yields $s' \leftrightarroweq \pi_n(s) \leftrightarroweq \pi_n(t) \leftrightarroweq t'$. Then, by completeness of the axiomatisation of ACP modulo bisimulation equivalence, $s' = t'$. Hence, $\pi_n(s) = s' = t' = \pi_n(t)$, that is, $\pi_n(\langle X_1\rangle) = \pi_n(\langle Y_1\rangle)$. By $\mathrm{AIP^c}$, we have that $\langle X_1\rangle = \langle Y_1\rangle$, i.e., $s = t$. □

## 6 Conclusion

We have reviewed $\mathrm{AIP^c}$ in the presence of the deadlock constant and have presented a soundness and completeness proof for the axiom system $\mathrm{ACP_{lin}} + \mathrm{PR} + \mathrm{AIP^c}$.

Our approach differs from Mauw's work in some main points. In order to derive the equality of two processes, Mauw states that the number of projections to be considered is equal to the sum of the number of linear equations used to define both processes. This way, the bound used for the number of projections is $n + m$, where $n$ and $m$ are the number of equations in each linear recursive specification. Our approach considers one linear recursive specification with $n$ equations. As a result of the deadlock constant and of the axiom PR2 ($\pi_0(x) = \delta$), the bound was reduced to $n - 1$.

Another difference between Mauw's work and ours is the model used. In the proof for the complete inference system, Mauw uses the model of finite graphs modulo bisimulation and proves the completeness of an axiom system called $\mathrm{BPA}_{\mathrm{Reg}}$ (BPA with regular processes). In our work, we use structural operational semantics and bisimulation equivalence and prove the completeness of the axiom system $\mathrm{ACP}_{\mathrm{lin}} + \mathrm{PR} + \mathrm{AIP}^{\mathrm{c}}$.

# Acknowledgements

We thank Inge Bethke, Alban Ponse and Mark van der Zwaag for drawing our attention to this work and for many useful suggestions.

# References

[1] Aceto, L., Fokkink, W., and Verhoef, C. Structural operational semantics. In *Handbook of Process Algebra* (2001), J. A. Bergstra, A. Ponse, and S. A. Smolka, Eds., Elsevier Science, pp. 197–292.

[2] Baeten, J. C. M., Bergstra, J. A., and Klop, J. W. On the consistency of koomen's fair abstraction rule. *Theoretical Computer Science 51(1/2) 51*, 1/2 (1987), 129–176.

[3] Baeten, J. C. M., and Weijland, W. P. *Process Algebra*, first ed. Cambridge University Press, 1990.

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

[5] de Bakker, J. W., and Zucker, J. I. Processes and the denotational semantics of concurrency. *Information and Control 54* (1982), 70–120.

[6] Fokkink, W. *Introduction to Process Algebra*, first ed. Springer, 2000.

[7] Mauw, S. A constructive version of the approximation induction principle. In *Proceedings . SION Conf. CSN 87* (1987), pp. 235–252.

[8] Milner, R. Calculi for synchrony and asynchrony. *Theoretical Computer Science 25* (1982), 267–310.

[9] Park, D. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science* (London, UK, 1981), Springer-Verlag, pp. 167–183.

[10] van Glabbeek, R. J. Bounded nondeterminism and the approximation induction principle in process algebra. Tech. Rep. CS-R8634, Centrum voor Wiskunde en Informatica (CWI), 1986.

## Electronic Reports Series of the Programming Research Group

Within this series the following reports appeared.

[PRG0801] J.A. Bergstra and C.A. Middelburg, *Programming an Interpreter Using Molecular Dynamics*, Programming Research Group - University of Amsterdam, 2008.

[PRG0713] J.A. Bergstra, A. Ponse, and M.B. van der Zwaag, *Tuplix Calculus*, Programming Research Group - University of Amsterdam, 2007.

[PRG0712] J.A. Bergstra, S. Nolst Trenite, and M.B. van der Zwaag, *Towards a Formalization of Budgets*, Programming Research Group - University of Amsterdam, 2007.

[PRG0711] J.A. Bergstra and C.A. Middelburg, *Program Algebra with a Jump-Shift Instruction*, Programming Research Group - University of Amsterdam, 2007.

[PRG0710] J.A. Bergstra and C.A. Middelburg, *Instruction Sequences with Dynamically Instantiated Instructions*, Programming Research Group - University of Amsterdam, 2007.

[PRG0709] J.A. Bergstra and C.A. Middelburg, *Instruction Sequences with Indirect Jumps*, Programming Research Group - University of Amsterdam, 2007.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[PRG0501] J.A. Bergstra and A. Ponse, *A Bypass of Cohen's Impossibility Result,* Programming Research Group - University of Amsterdam, 2005.

[PRG0405] J.A. Bergstra and I. Bethke, *An Upper Bound for the Equational Specification of Finite State Services,* Programming Research Group - University of Amsterdam, 2004.

[PRG0404] J.A. Bergstra and C.A. Middelburg, *Thread Algebra for Strategic Interleaving,* Programming Research Group - University of Amsterdam, 2004.

[PRG0403] B. Diertens, *A Compiler-projection from PGLEc.MSPio to Parrot,* Programming Research Group - University of Amsterdam, 2004.

[PRG0402] J.A. Bergstra and I. Bethke, *Linear Projective Program Syntax,* Programming Research Group - University of Amsterdam, 2004.

[PRG0401] B. Diertens, *Molecular Scripting Primitives,* Programming Research Group - University of Amsterdam, 2004.

[PRG0302] B. Diertens, *A Toolset for PGA,* Programming Research Group - University of Amsterdam, 2003.

[PRG0301] J.A. Bergstra and P. Walters, *Projection Semantics for Multi-File Programs,* Programming Research Group - University of Amsterdam, 2003.

[PRG0201] I. Bethke and P. Walters, *Molecule-oriented Java Programs for Cyclic Sequences,* Programming Research Group - University of Amsterdam, 2002.

The above reports and more are available through the website: www.science.uva.nl/research/prog/

Electronic Report Series

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ  Amsterdam
the Netherlands

www.science.uva.nl/research/prog/