Bounded Nondeterminism

and the Approximation Induction Principle in Process Algebra

(Extended Abstract)

R.J. van Glabbeek

Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

This paper presents a new semantics of ACP_τ, the Algebra of Communicating Processes with abstraction. This leads to a term model of ACP_τ which is isomorphic to the model of process graphs modulo rooted $\tau\delta$-bisimulation of BAETEN, BERGSTRA & KLOP [2], but in which no special rootedness condition is needed. Bisimilarity turns out to be a congruence in a natural way.

In this model, the Recursive Definition Principle (RDP), the Commutativity of Abstraction (CA) and Koomen's Fair Abstraction Rule (KFAR) are satisfied, but the Approximation Induction Principle (AIP) is not. The combination of these four principles is proven to be inconsistent, while any combination of three of them is not.

In [2] a restricted version of AIP is proved valid in the graph model. This paper proposes a simpler and less restrictive version of AIP, not containing guarded recursive specifications as a parameter, which is still valid. This infinitary rule is formulated with the help of a family B_n of unary predicates, expressing bounded nondeterminism.

1980 Mathematics Subject Classification (version 1985): 68Q10, 68Q45, 68Q55, 68N15.
1982 CR Categories: F.1.2, F.3.2, F.4.3, D.3.1.
Key Words & Phrases: Concurrency, Process algebra, ACP, Approximation Induction Principle, Recursion, Abstraction, Fairness, Liveness, Consistency, Bisimulation, Bounded Nondeterminism.
Note: Sponsored in part by Esprit project no. 432, METEOR.

INTRODUCTION

Concurrency

A process is the behaviour of a system. The system can be a machine, a communication protocol, a network of falling dominoes, a chess player, or any other system. Concurrency is the study of parallel processes. The features studied include communication between parallel processes, deadlock behaviour, abstraction from internal steps, divergence, nondeterminism, fairness, priorities in the choice of actions, tight regions, etc. Processes are mostly studied within a model, capturing some of the features of concurrency. Among these models one finds Petri nets (see for instance REISIG [13]), Topological models (as in DE BAKKER & ZUCKER [3]), Algebraical models (like the projective limit models in BERGSTRA & KLOP [4]), Graph models (as in MILNER [10] and in BAETEN, BERGSTRA & KLOP [2]) and observation models, in which a process is fully determined by its possible interactions with the environment (like Hoare's failures model of Communicating Sequential Processes, see BROOKES, HOARE & ROSCOE [7], and the models used in Trace theory, see for instance REM [14]). Parameters in the classification of these models of concurrency are the features captured by the model, the identifications made on processes and the particular way of representing them. The identification issue deals with the question when two processes are to be considered equal. This is of importance on judging whether or not a certain system correctly implements a specification. The possible answers constitute a broad spectrum of process semantics, ranging from trace semantics, where two processes are identified as soon as their possible sequences of actions coincide, to bisimulation semantics, where all information about the timing of the divergencies of those traces is preserved.

Process algebra

Process algebra is an algebraic approach to the study of concurrent processes. Its tools are algebraical languages for the specification of processes and the formulation of statements about them, together with calculi for the verification of these statements. Process algebra is not to be regarded as a model of concurrency. On the one hand it is a method for specifying processes and proving statements about them without being limited to a particular model; on the other hand it is a method for analysing and comparing the different models of concurrency.

To illustrate the first application, consider a typical example. Suppose a machine is composed out of two components. In order to verify that it behaves as it should, one specifies the behaviour of the two components as well as the intended behaviour of their composition in an algebraical language. This language should be equipped with a composition operator and with a calculus, consisting of laws concerning the equality relation, the composition operator and the operators involved in the specifications of the three processes. In selecting the calculus it should be checked that all its rules and axioms are valid in the environment in which the machine is operating. Now one is able to formulate and prove the statement: the behaviour of the composition of the two components is equal to the intended behaviour of the desired machine.

The creation of an algebraical framework suitable to deal with such applications, gives rise to the construction of

building blocks of operators and axioms, each block describing a feature of concurrency in a certain semantical setting. The models of concurrency serve to prove the consistency of the theories built from these blocks, and to illustrate the range of their applicability.

As to the second application, the various models of concurrency can be studied and classified by axiomatising them, and pointing out which axioms constitute the differences between them.

The first axiomatic treatment of concurrency is Milner's Calculus of Communicating Systems [10]. This calculus is closely linked to Milner's graph model (of 'synchronisation trees') with bisimulation semantics, and the axioms are presented as theorems, valid in this model. Other calculi are Milne's CIRCAL [9] and the Algebra of Communicating Processes (ACP) of BERGSTRA & KLOP [4]. The last one is not tied to a particular model. It is the core of a family of axioms systems, fitting in the process algebra methodology sketched above. Its standard semantics is bisimulation semantics, since it identifies the least; any theorem proved in bisimulation semantics remains valid in coarser semantics; but there are building blocks with axioms for more identifications. The present paper examines some rules and axioms, belonging to this family, and employs the notation of ACP. Although it builds further on the research done in [4] and [2], it can be read independently. For further details is refered to the full paper [8].

1. ATOMIC ACTIONS AND COMMUNICATIONS

An atomic action is the most elementary component of a process. It is considered not to be divisible into smaller parts and not subject to further investigations. Mostly an atomic action is considered to be observed pointwise in time, for if the time it takes is to be observed, two atomic actions can be distinguished: its beginning and its end. It depends on the level of abstraction, which actions one wants to see as atomic.

Atomic actions are thought to occur simultaneously in a process only if they are communicating, like the actions 'give' and 'receive'. The simultaneous occurrence of actions a and b is denoted by $a|b$. In general $a|b = b|a$ and $(a|b)|c = a|(b|c)$. A multiset $a_1|\cdots|a_n$ (with $n\geqslant 2$) of communicating atomic actions is called a communication. The presentation of an algebra of communicating processes starts with postulating an alphabet A^0 of atomic actions and specifying which communications can occur.

Formally, an alphabet A of atomic actions and communications is defined as a set of nonempty multisets of symbols, such that if $a\in A$ and $b\subseteq a$ then also $b\in A$. Elements of A are called *actions*. A singleton action is called *atomic*; other actions are *communications*. A^0 is the set of atomic actions in A. Two actions a and $b\in A$ are said to communicate if their union $a|b\in A$.

Example: $A = \{a, b, c, b|c, c|c, b|c|c\}$. There is communication possible between b and c, c and c, $b|c$ and c and between b and $c|c$, while there is no communication possible between a and b or between b and $b|c$.

If $A = \{a, b, a|b\}$ and one wants to use c as an abbreviation for $a|b$, write $A = \{a, b, a|b=c\}$. This presentation differs slightly from the presentation in BERGSTRA & KLOP [4,5], where A contains only atomic actions and communication is given by a partial binary function $|: A\times A\rightarrow A$. There the last example would be $A = \{a,b,c\}$ and $a|b = c$.

2. THE ALGEBRA OF COMMUNICATING PROCESSES WITH ABSTRACTION

ACP_τ, the algebra of communicating processes with abstraction, is the equational theory, presented in the upper blocks of table 2. Its language is built inductively from a set $V = \{x,y,z,\ldots\}$ of variables, and the constants and operators of table 1. The equality predicate $=$ is always present, but never mentioned. An alphabet A of atomic actions and communications occurs as a parameter in ACP_τ.

ACP_τ:	constants:	a	for any atomic action $a\in A^0$
		δ	deadlock
		τ	silent action
	unary operators:	∂_H	encapsulation, for any $H\subseteq A$
		τ_I	abstraction, for any $I\subseteq A$
	binary operators:	$+$	alternative composition (sum)
		$\cdot$	sequential composition (product)
		$\|$	parallel composition (merge)
		$\lfloor$	left-merge
		$\mid$	communication merge (bar)

Table 1

The meaning of these constructs will be given informally below, together with an explanation of the axioms of ACP_τ. In table 2, all axioms are in fact axiom schemes in a,b and c, with a,b,c ranging over $A\cup\{\delta\}$, unless further restrictions are made in the table.

ACP$_\tau$

$x+y = y+x$	A1	$x\tau = x$	T1
$x+(y+z) = (x+y)+z$	A2	$\tau x+x = \tau x$	T2
$x+x = x$	A3	$a(\tau x+y) = a(\tau x+y)+ax$	T3
$(x+y)z = xz+yz$	A4		
$(xy)z = x(yz)$	A5		
$x+\delta = x$	A6		
$\delta x = \delta$	A7		
$a\|b = b\|a$	C1		
$(a\|b)\|c = a\|(b\|c)$	C2		
$a\|b = \delta$ if $a\|b \notin A$	C3		
$x\|\|y = x\lfloor\lfloor y+y\lfloor\lfloor x+x\|y$	CM1		
$a\lfloor\lfloor x = ax$	CM2	$\tau\lfloor\lfloor x = \tau x$	TM1
$(ax)\lfloor\lfloor y = a(x\|\|y)$	CM3	$(\tau x)\lfloor\lfloor y = \tau(x\|\|y)$	TM2
$(x+y)\lfloor\lfloor z = x\lfloor\lfloor z+y\lfloor\lfloor z$	CM4	$\tau\|x = \delta$	TC1
$(ax)\|b = (a\|b)x$	CM5	$x\|\tau = \delta$	TC2
$a\|(bx) = (a\|b)x$	CM6	$(\tau x)\|y = x\|y$	TC3
$(ax)\|(by) = (a\|b)(x\|\|y)$	CM7	$x\|(\tau y) = x\|y$	TC4
$(x+y)\|z = x\|z+y\|z$	CM8		
$x\|(y+z) = x\|y+x\|z$	CM9		
		$\partial_H(\tau) = \tau$	DT
		$\tau_I(\tau) = \tau$	TI1
$\partial_H(a) = a$ if $a \notin H$	D1	$\tau_I(a) = a$ if $a \notin I$	TI2
$\partial_H(a) = \delta$ if $a \in H$	D2	$\tau_I(a) = \tau$ if $a \in I$	TI3
$\partial_H(x+y) = \partial_H(x)+\partial_H(y)$	D3	$\tau_I(x+y) = \tau_I(x)+\tau_I(y)$	TI4
$\partial_H(xy) = \partial_H(x)\cdot\partial_H(y)$	D4	$\tau_I(xy) = \tau_I(x)\cdot\tau_I(y)$	TI5

PR

$$\pi_n(\tau) = \tau$$
$$\pi_0(ax) = \delta$$
$$\pi_{n+1}(ax) = a\cdot\pi_n(x)$$
$$\pi_n(\tau x) = \tau\cdot\pi_n(x)$$
$$\pi_n(x+y) = \pi_n(x)+\pi_n(y)$$

B

$$B_0(x) \quad B_n(a) \quad B_n(\tau) \quad \frac{B_n(x)}{B_n(\tau x)} \quad \frac{B_n(x)}{B_{n+1}(ax)} \quad \frac{B_n(x), B_n(y)}{B_n(x+y)}$$

AIP$^-$

$$\frac{\forall n \in \mathbb{N}\ \pi_n(x)=\pi_n(y), B_n(x)}{x=y}$$

KFAR

$$\frac{x = ix+y}{\tau_{\{i\}}(x)=\tau\cdot\tau_{\{i\}}(y)}$$

CA

$$\tau_I\circ\tau_J(x)=\tau_{I\cup J}(x)$$

Table 2

a represents the process, starting with an a-step and terminating after some time. (see below)

δ is the action of acknowledging that there is no possibility to proceed. Put $A_\delta = A \cup \{\delta\}$.

τ represents the process terminating after some time, without performing observable actions. Put $A_\tau = A \cup \{\tau\}$.

$x+y$ represents the process that executes either x or y. The choice between x and y is made at the begining of $x+y$. It is not specified by whom. The axioms A1, A2 and A3 state that in a choice the alternatives are regarded to form a set. Axiom A6 states that deadlock only occurs if there are no alternatives.

$x\cdot y$ represents the process x, followed after possible termination by y. The process x fails to terminate if it ends in deadlock (A7), or if it performs an infinite sequence of actions, or if it goes on forever without performing any action. The last possibility is called *divergence*. The axioms A4 and A5 are rather straightforward, but

since (at least in bisimulation semantics) the timing of the choices is of importance, there is no axiom $x(y+z) = xy+xz$.

$x \| y$ represents the simultaneous execution of x and y. It starts when one of its components starts and terminates if both of them do. (see below)

$x \lfloor\!\lfloor y$ is as $x \| y$, but under the assumption that x starts first (CM2,3,4, TM1,2).

$x \mid y$ is as $x \| y$, but starting with a communication between x and y (CM5,6,7,8,9). This communication may be preceded by some silent steps, but these are no part of the process (TC3,4). Silent processes do not take part in communications (TC1,2). Axiom CM1 states that a process $x \| y$ starts either with x or with y or with a communication between x and y. If the first actions from x and y do not communicate (as is always the case if $x = \delta$ or $y = \delta$) the summand $x \mid y$ can be removed, using C3 and A6,7.

$\partial_H(x)$ represents the process x without the possibility of performing actions from H. ∂_H renames the actions from H into δ (DT, D1-4). Mostly it is used to remove the remnants of unsuccessful communication from a merge, thereby indicating that the process is not at the same time communicating (through H at least) with the environment. This is why ∂_H is called encapsulation.
Example: $A = \{give, receive, give \mid receive\}$; $H = \{give, receive\}$.

$$\partial_H(give \| receive) = \partial_H(give \cdot receive + receive \cdot give + give \mid receive) =$$
$$= \delta \cdot \delta + \delta \cdot \delta + give \mid receive = give \mid receive.$$

$\tau_I(x)$ represents the process x, of which the actions from I are not considered important anymore. τ_I renames the actions from I into τ (TI 1-5).

For $a \in A$ the chosen semantics of the expression a can be motivated as follows: a denotes a process executing an action a. Although both the process and the action are denoted by a, they are different entities. We require the process a to take a positive amount of time, since it seems a natural assumption that all activity takes some time. However, in this time interval there should be only one single point where the execution of the action a is recorded (one could take the first moment that evidence is available for the execution of a), for if we would allow an action to manifest itself during a positive time span, then equations as $a \| b = ab + ba$, which hold in ACP_τ, could be refuted on grounds of real-time behaviour. Hence there are three events to be recorded in the life of the process a: its beginning, the occurrence of the action a and its termination. Now the question arises where the second event has to be situated between the other ones. Suppose a process like $a+b$ operates in an environment where a cannot be executed (the process expression $a+b$ appears in the scope of a $\partial_{\{a\}}$ operator); then this option cancels out and the process will perform a b-step. However, if the process a can start without being recognizable as the process a, then it will be too late to do a b-step if the action a turns out to be impossible, and deadlock occurs. So, in order to define a ∂_H operator properly, one has to assume that the identity of a process a is clear from its beginning, at least for an environment $\partial_H(\cdot)$. Therefore the expression a is chosen to denote a process that starts with executing the action a. A process that does the a-step only after some time can be denoted by the expression τa (see figure 1).

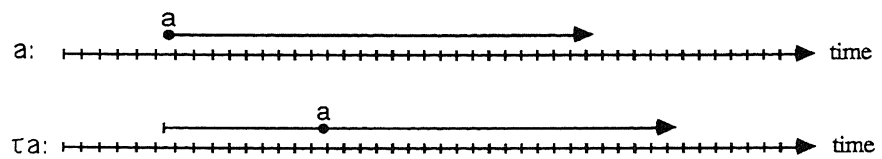

fig. 1

Now it is possible to motivate the τ-laws of table 2.

T1 follows since any ACP_τ-process terminates only after a period in which no actions are performed. Since the length of this period is not specified, the process can be identified with a process that waits a little longer before terminating. A law $\tau x = x$ can not be defended in the same way. a for instance is a process that has to start with an a-step, while τa may wait some time first.

T2 is less obvious in this semantics. τa is a process that has to wait some time before the a-step can be performed, while $\tau a + a$ has the option to perform an a-step immediately. However, in ACP_τ these processes are identified, since the property 'having to wait some time before an action can happen' is not considered important enough for discriminating between processes. This in contrast with the property 'being able to wait some time before an action happens', which *is* used to discriminate between processes: $\tau a + a \neq a$. The argument is that a $\partial_H(\cdot)$ environment (in which certain actions can not be executed) discriminates only on grounds of the second property: in a sum only those summands can be canceled that are not able to wait some time before an action happens.
Of course it is also possible to use a more subtle semantics in which both properties are used for distinguishing between processes. However, in unrestricted form such a semantics would clash with the axioms CM2

and CM3: $ab = a \mathbin{\lfloor\!\lfloor} b = (a\tau) \mathbin{\lfloor\!\lfloor} b = a(\tau b + b)$. Therefore this option is not pursued in this paper.

T3 is adopted since $a(\tau x + y)$ already has the possibility to start with an a-step and then, after some time, reach a state where only x is possible.

These arguments will be formalised in section 5. Then propositions 4 and 5 of section 9 will tell us that the τ-laws exactly reflect the proposed semantics in the theory.

Using the axioms of table 2 it turns out that $\tau_{\{a\}}(a \| b)$ is equal to τb and hence different from b:

$$\tau_{\{a\}}(a \| b) = \tau_{\{a\}}(ab + ba) = \tau b + b\tau = \tau b + b = \tau b.$$

This fits in with the meaning of $x \| y$, given in section 2. $a \| b$ starts when one of its components starts and terminates if both of them do.

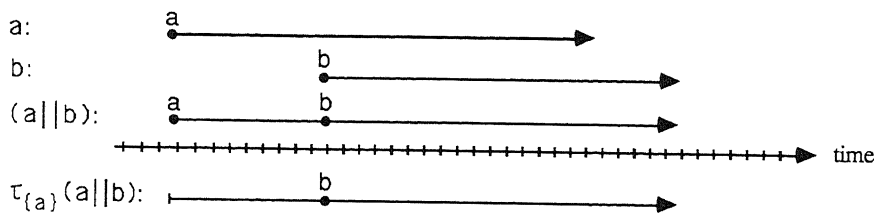

fig. 2

As indicated in figure 2, it is quite possible that the process a starts before b does. After abstraction from a, the resulting process is τb.

ACP_τ was first presented in BERGSTRA & KLOP [5]. In this presentation only axiom C3 is different, as a consequence of the different treatment of communication, mentioned in section 1. The τ-laws T1,T2 and T3 originate from MILNER [10]. In [5] it is proved that ACP_τ is a sound and complete proof system for closed recursion free terms, with respect to the semantical notion of bisimulation. However it is possible to make more identifications (depending on a notion of observability for instance), by adding some axioms. In a completeness proof it is important to know that any finite closed process expression can be rewritten into a process expression, built up inductively, following the scheme τ, ax, τx, $x + y$. Finally note that the axioms CM2,5 and 6 are derivable from the others.

3. RECURSION

A *recursive specification* E is a set of equations $\{x = t_x \mid x \in V_E\}$ with V_E a set of variables and t_x an ACP_τ term for $x \in V_E$. The variables of V_E may appear in t_x. Other variables occurring in t_x ($x \in V_E$) are called *parameters* of E. Mostly, only recursive specifications without parameters are used. A *solution* of E is an interpretation of the variables of V_E as processes (in a certain domain) (as a function of an interpretation of the parameters of E), such that the equations of E are satisfied.

The Recursive Definition Principle (RDP) tells us that every recursive specification has a solution. In section 9 a model for ACP_τ will be presented, satisfying RDP. RDP cannot be expressed algebraically, since in algebraic languages no existential quantification is permitted.

Recursive specifications are used to define (or *specify*) processes. If E has a unique solution, let $\langle x \mid E \rangle$ (with $x \in V_E$) denote the x-component of this solution. If E has more than one solution, $\langle x \mid E \rangle$ denotes 'one of the solutions of E', and can be regarded as a kind of variable, ranging over these solutions. If E has no solutions (possible in a model, not satisfying RDP), then no meaning can be attached to $\langle x \mid E \rangle$. In a recursive language, the syntactical constructs $\langle x \mid E \rangle$ may appear in the construction of terms (possibly nested). This limits the class of models of the language to the ones satisfying RDP.

In most applications the variables $X \in V_E$ in a recursive specification E will be chosen freshly, so that there is no need to repeat E in each occurrence of $\langle X \mid E \rangle$. Therefore the convention will be adopted that once a recursive specifications is declared, $\langle X \mid E \rangle$ can be abbreviated by X. If this is done, X is called a *formal variable*. Formal variables are denoted by capital letters. So after the declaration $X = aX$, a statement $X = aaX$ should be interpreted as an abbreviation of $\langle X \mid X = aX \rangle = aa \langle X \mid X = aX \rangle$.

Let $E = \{x = t_x \mid x \in V_E\}$ be a recursive specification, and t a process expression. Then $\langle t \mid E \rangle$ denotes the term t in which each (free) occurrence of $x \in V_E$ is replaced by $\langle x \mid E \rangle$ (avoiding name clashes). In a recursive language all formulas $\langle x \mid E \rangle = \langle t_x \mid E \rangle$ (with E as above and $x \in V_E$) may be considered provable. If the above convention is used, these formulas seem to be just the equations of E.

Let T be an equational theory (like ACP_τ) over a signature Σ and e a Σ-equation, both recursion free. The following

notation is employed:
$T \vdash e$: e is provable from T in the recursion free language over Σ.
$T \models e$: e is true in all Σ-algebras, satisfying T.
$T+RDP \vdash e$: e is provable from T in the recursive language over Σ.
$T+RDP \models e$: e is true in all Σ-algebras, satisfying T and RDP.
Now Birkhof's completeness theorem for equational logic reads:

$$T \vdash e \Leftrightarrow T \models e.$$

It can be extended trivially to the case where T contains also conditional equations, and predicates are allowed in Σ. Now the following proposition, whose proof is omitted, justifies the notation $T+RDP \vdash e$.

PROPOSITION 1. $T+RDP \vdash e \Leftrightarrow T+RDP \models e$.

4. THE APPROXIMATION INDUCTION PRINCIPLE

PR is a building block that can be added to ACP_τ. It consists of the unary operators π_n (projection) for $n \in \mathbb{N}$, and the axioms presented in the third block of table 2. $\pi_n(x)$ represents the process x, which is only allowed to perform n visible actions. The next visible action is blocked, i.e. renamed into δ.

The Approximation Induction Principle (AIP) is the infinitary rule

$$\text{AIP} \quad \frac{\forall n \in \mathbb{N}\ \pi_n(x)=\pi_n(y)}{x=y}$$

saying that a process is fully determined by its finite projections. It follows if one choses to identify processes that can not be distinguished by their finite observations.

5. ACTION RELATIONS BETWEEN PROCESSES

If x and y are processes and $a \in A$, then write
$x \xrightarrow{a} y$ if x can evolve into y during a period in which only the action a is performed.
$x \xrightarrow{\tau} y$ if x can evolve into y, taking a positive amount of time in which no visible actions occur.
$x \xrightarrow{a} \surd$ if x can terminate after having done only an a-step, and
$x \xrightarrow{\tau} \surd$ if x can terminate after some time, without performing visible actions.
The involved binary predicates $\xrightarrow{a}$ and unary predicates $\xrightarrow{a}\surd$, both for $a \in A_\tau$, are called *action relations*. A *process expression* is a term in the recursive language of $ACP_\tau+PR$. Let $\mathcal{P}$ be the domain of closed process expressions. On $\mathcal{P}$ the action relations are generated by the *action rules* presented in the upper block of table 3. All these rules are schemes in a and b, with a,b ranging over A_τ, unless further restrictions are made in the table. These action rules provide an alternative explanation of the meaning of the language constructs of $ACP_\tau+PR+RDP$. Note that the τ-laws of table 3 exactly reflect the motivations for the τ-laws of table 2, as given in section 2, in terms of action relations. Also note that the instance $\tau \xrightarrow{\tau} \tau$ of the first τ-law of table 3 does not imply that the process τ may fail to terminate. It only says that after waiting some time on the process to terminate, it may still not be ready. The situation is as in Zeno's paradox: Achilles *will* catch up with the Tortoise.

The action relations can be generalised as follows. If x and y are processes and $\sigma \in A^*$, then write

$x \xrightarrow{\sigma\gg} y$ if x can evolve into y during a period (of positive duration) in which (only) the sequence of actions σ occurs.

$x \xrightarrow{\sigma\gg} \surd$ if x can terminate after a period (of positive duration) in which (only) the sequence of actions σ occurs.
On $\mathcal{P}$ these generalised action relations are generated by the rules presented in the bottom block of table 3. There $\tau \in A^*$ denotes the empty string, a ranges over A and σ,ρ over A^*.

6. FAIR ABSTRACTION

EXAMPLE (due to F. Vaandrager): A statistician performs a simple experiment in a closed room: he tosses a coin until tail comes up; then he leaves the room to report success. Let p be the probability that, if he tosses the coin, tail comes up. Assume $0<p<1$. The behaviour of the statistician is specified by

$$S = head \cdot S + tail \cdot success$$

Being outside the room, the only part of the process we can observe is the statistician leaving the room to report

a :	$a \xrightarrow{a} \surd$				
$+$:	$\dfrac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'}$	$\dfrac{x \xrightarrow{a} \surd}{x+y \xrightarrow{a} \surd}$	$\dfrac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}$	$\dfrac{y \xrightarrow{a} \surd}{x+y \xrightarrow{a} \surd}$	
$\cdot$:	$\dfrac{x \xrightarrow{a} x'}{xy \xrightarrow{a} x'y}$	$\dfrac{x \xrightarrow{a} \surd}{xy \xrightarrow{a} y}$			
$\|$:	$\dfrac{x \xrightarrow{a} x'}{x\|y \xrightarrow{a} x'\|y}$	$\dfrac{x \xrightarrow{a} \surd}{x\|y \xrightarrow{a} y}$	$\dfrac{y \xrightarrow{a} y'}{x\|y \xrightarrow{a} x\|y'}$	$\dfrac{y \xrightarrow{a} \surd}{x\|y \xrightarrow{a} x}$	
	$\dfrac{x \xrightarrow{a} x', y \xrightarrow{b} y'}{x\|y \xrightarrow{a\mid b} x'\|y'}$	$\dfrac{x \xrightarrow{a} \surd, y \xrightarrow{b} y'}{x\|y \xrightarrow{a\mid b} y'}$	$\dfrac{x \xrightarrow{a} x', y \xrightarrow{b} \surd}{x\|y \xrightarrow{a\mid b} x'}$	$\dfrac{x \xrightarrow{a} \surd, y \xrightarrow{b} \surd}{x\|y \xrightarrow{a\mid b} \surd}$	(if $a \mid b \in A$)
$\lfloor$:	$\dfrac{x \xrightarrow{a} x'}{x \lfloor y \xrightarrow{a} x\|y}$	$\dfrac{x \xrightarrow{a} \surd}{x \lfloor y \xrightarrow{a} y}$			
$\mid$:	$\dfrac{x \xrightarrow{a} x', y \xrightarrow{b} y'}{x \mid y \xrightarrow{a\mid b} x'\|y'}$	$\dfrac{x \xrightarrow{a} \surd, y \xrightarrow{b} y'}{x \mid y \xrightarrow{a\mid b} y'}$	$\dfrac{x \xrightarrow{a} x', y \xrightarrow{b} \surd}{x \mid y \xrightarrow{a\mid b} x'}$	$\dfrac{x \xrightarrow{a} \surd, y \xrightarrow{b} \surd}{x \mid y \xrightarrow{a\mid b} \surd}$	(if $a \mid b \in A$)
∂_H:	$\dfrac{x \xrightarrow{a} x'}{\partial_H(x) \xrightarrow{a} \partial_H(x')}$	$\dfrac{x \xrightarrow{a} \surd}{\partial_H(x) \xrightarrow{a} \surd}$	(if $a \notin H$)		
τ_I:	$\dfrac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{a} \tau_I(x')}$	$\dfrac{x \xrightarrow{a} \surd}{\tau_I(x) \xrightarrow{a} \surd}$	(if $a \notin I$)		
	$\dfrac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}$	$\dfrac{x \xrightarrow{a} \surd}{\tau_I(x) \xrightarrow{\tau} \surd}$	(if $a \in I$)		
π_n:	$\dfrac{x \xrightarrow{a} x'}{\pi_{n+1}(x) \xrightarrow{a} \pi_n(x')}$	$\dfrac{x \xrightarrow{a} \surd}{\pi_{n+1}(x) \xrightarrow{a} \surd}$	(if $a \neq \tau$)		
	$\dfrac{x \xrightarrow{\tau} x'}{\pi_n(x) \xrightarrow{\tau} \pi_n(x')}$	$\dfrac{x \xrightarrow{\tau} \surd}{\pi_n(x) \xrightarrow{\tau} \surd}$			
recursion:	$\dfrac{<t_x \mid E> \xrightarrow{a} y}{<x \mid E> \xrightarrow{a} y}$	$\dfrac{<t_x \mid E> \xrightarrow{a} \surd}{<x \mid E> \xrightarrow{a} \surd}$			
τ - laws:	$a \xrightarrow{a} \tau$	$\dfrac{x \xrightarrow{\tau} y,\ y \xrightarrow{a} z}{x \xrightarrow{a} z}$	$\dfrac{x \xrightarrow{\tau} y,\ y \xrightarrow{a} \surd}{x \xrightarrow{a} \surd}$	$\dfrac{x \xrightarrow{a} y,\ y \xrightarrow{\tau} z}{x \xrightarrow{a} z}$	$\dfrac{x \xrightarrow{a} y,\ y \xrightarrow{\tau} \surd}{x \xrightarrow{a} \surd}$

$\dfrac{x \xrightarrow{a} y}{x \xrightarrow{a}\!\!\gg y}$	$\dfrac{x \xrightarrow{a} \surd}{x \xrightarrow{a}\!\!\gg \surd}$	$\dfrac{x \xrightarrow{\tau} y}{x \xrightarrow{\tau}\!\!\gg y}$	$\dfrac{x \xrightarrow{\tau} \surd}{x \xrightarrow{\tau}\!\!\gg \surd}$	$\dfrac{x \xrightarrow{\sigma}\!\!\gg y,\ y \xrightarrow{\rho}\!\!\gg z}{x \xrightarrow{\sigma\rho}\!\!\gg z}$	$\dfrac{x \xrightarrow{\sigma}\!\!\gg y,\ y \xrightarrow{\rho}\!\!\gg \surd}{x \xrightarrow{\sigma\rho}\!\!\gg \surd}$

Table 3

success. So the actions from $I = \{head, tail\}$ are hidden, and the observed process is $\tau_I(S)$. Since $0<p<1$, the process S will perform a *tail* action sooner or later, which yields the identity

$$\tau_I(S) = \tau \cdot success.$$

What is needed is an algebraic framework in which one can prove this equation.

An infinite path of a process x is an infinite alternating sequence of labels $a_i \in A_\tau$ and processes x_i ($i \in \mathbb{N}$), such that $x \xrightarrow{a_0} x_0 \xrightarrow{a_1} x_1 \xrightarrow{a_2} x_2 \rightarrow \ldots$.

Such a path has an *exit* at $x_i (i \in \mathbb{N})$ if $x_i \xrightarrow{b} y$ with either $b \neq a_{i+1}$ or $y \neq x_{i+1}$. This exit is called a τ-*exit* if $b = \tau$. A path is called *improbable* if it has infinitely many exits. Now a process is said to be *fair,* if for any improbable path the probability that it will be executed is zero. In a theory for fair processes there is room for proof rules stating that certain improbable paths may be discarded. There is however a problem in discarding improbable paths. If a process is placed in a context $\partial_H(\cdot)$ then certain paths may stop to be improbable because their exits disappear. In that case they may not have been discarded. Thus only paths may be discarded which are improbable in all contexts. These are the paths with infinitely many τ-exits. KFAR$^-$ is a proof rule, stating that certain paths with infinitely many τ-exits, which are made invisible by a τ_I operator, may be discarded.

$$\text{KFAR}^- \quad \frac{x = ix + \tau y + z}{\tau_{\{i\}}(x) = \tau \cdot \tau_{\{i\}}(\tau y + z)}$$

A version of KFAR$^-$ appeared first in BERGSTRA, KLOP & OLDEROG [6]. It is a restricted version of Koomen's Fair Abstraction Rule (KFAR), which was presented in BAETEN, BERGSTRA & KLOP [2], and will be discussed in the next section.

The last axiom of table 2, the commutativity of abstraction (CA), says that it does not matter in which order actions are considered unimportant (or made invisible). It occurred already as one of the conditional axioms (also abbreviated as CA) in BAETEN, BERGSTRA & KLOP [1], and will play an important role in observations to come.

Using KFAR$^-$ and CA the identity $\tau_I(S) = \tau \cdot success$ from the example of the statistician can be derived formally:

$$\tau_{\{tail\}}(S) = \tau_{\{tail\}}(head \cdot S + tail \cdot success) = head \cdot \tau_{\{tail\}}(S) + \tau \cdot success + \delta,$$

$$\text{so } \tau_I(S) = \tau_{\{head\}} \circ \tau_{\{tail\}}(S) = \tau \cdot \tau_{\{head\}}(\tau \cdot success + \delta) = \tau \cdot success.$$

A theory, containing rules like KFAR$^-$ is only suited for the study of fair processes. For any application it has to be checked that all processes concerned are fair indeed.

7. DEADLOCK = LIVELOCK

EXAMPLE. Choose $A = \{a, b, c, b|c\}$ and $H = \{b, c\}$. Then $\text{ACP}_\tau \vdash \partial_H(aaab \| c) = aaa(b|c)$. So the process c inside the encapsulated merge $\partial_H(aaab \| \cdot)$ waits patiently until it can communicate with *aaab*. If such a communication is not possible, deadlock occurs:

$$\text{ACP}_\tau \vdash \partial_H(aaa \| c) = aaa\delta$$

$$\text{ACP}_\tau \vdash \partial_H(aaac \| c) = aaa\delta.$$

So deadlock occurs in an encapsulated merge if not all components are terminated, and the ones which are not are all waiting for an opportunity to communicate. From this one learns that deadlock, as in $aaa\delta$, should not be interpreted as a violent crash of the system, but as an eternal sleep.

EXAMPLE. Specify X by $X = aX$. Then $\tau_{\{a\}}(X)$ remains active forever (it performs a-steps), but no actions can be observed. This is called *livelock.*

In order to distinguish deadlock from livelock one can assume that processes are noisy. The noise starts at the beginning of a process, and ends if the process terminates or starts waiting. If a component in an encapsulated merge has to wait for a suitable communication it becomes silent until the communication is enabled, but as long as at least one component is making progress (visibly or invisibly) noise is being made. Only if all components are waiting (or terminated), the process becomes silent. This guarantees that no further action is possible and it will remain silent forever. In such a semantics, deadlock is observable (silence is), but livelock is not (one can never know that from some moment on no visible action will be performed). In bisimulation semantics, as employed in this paper, processes are not assumed to be noisy, and no distinction between deadlock and livelock is made. This can be expressed algebraically by the rule:

$$\text{deadlock} = \text{livelock} \quad \frac{x = ix}{\tau_{\{i\}}(x) = \tau\delta}$$

In this rule livelock is expressed as $\tau_{\{i\}}(x)$, with x satisfying $x=ix$, and deadlock as $\tau\delta$. Note that livelock can not be expressed as a process x satisfying $x=\tau x$, since also τa satisfies $x=\tau x$. Furthermore deadlock can not be expressed by δ, since in $a\cdot(b+\delta)$ no deadlock occurs.

Equating deadlock and livelock amounts to stating that in a process invisible infinite paths without any exits may be discarded (leaving $\tau\delta$ in place). In combination with fairness this means that any invisible infinite path may be discarded, regardless whether it is improbable or not. This is expressed by Koomen's Fair Abstraction Rule:

$$\text{KFAR}\quad \frac{x=ix+y}{\tau_{\{i\}}(x)=\tau\cdot\tau_{\{i\}}(y)}$$

The rule *deadlock = livelock* can be obtained from KFAR by substituting $y=\delta$. KFAR$^-$ can be obtained by substituting $y=\tau y+z$.

8. Consistency & Liveness

For $p\in\mathcal{P}$ write $p\not\rightarrow$ if $p\xrightarrow{a}q$ for no $a\in A_\tau$ and $q\in\mathcal{P}$. A theory T is said to be *consistent* if $T\vdash p=q$ implies:

(i) $p\xrightarrow{\sigma}\surd$ if and only if $q\xrightarrow{\sigma}\surd$

(ii) $p\xrightarrow{\sigma}p'\not\rightarrow$ if and only if $q\xrightarrow{\sigma}q'\not\rightarrow$

for any pair of closed recursion free process expressions p and q. This notion was called 'trace consistency' in Bergstra, Klop and Olderog [6]. A theory T with $T\vdash\tau=\tau+\tau\delta$ for instance is inconsistent. Depending on whether or not one wants to assume fairness and/or *deadlock = livelock* there are several ways to define consistency in terms of recursive process expressions as well. The main possibilities are discussed in Van Glabbeek [8]. In order to leave all options open, in this paper consistency is defined in terms of recursion free process expressions only. In this paper process theories are required to be consistent, so that deadlock behaviour can be dealt with properly.

The notions of safety and liveness are frequently used in the literature. Roughly, safety means that something bad cannot happen, while liveness means that something good will eventually happen. Use τ^ω as an abbreviation of $\tau_{\{i\}}(<x\mid x=ix>)$. If termination is considered to be something good then $T\vdash\tau=\tau+\tau^\omega$ should be regarded as a violation of liveness (by the theory T). In [8] this concept has been formalised.

9. A Term Model for ACP$_\tau$+PR+RDP+CA+KFAR.

A *bisimulation* is a binary relation R on $\mathcal{P}$, satisfying:

- if pRq and $p\xrightarrow{a}p'$, then $\exists q':q\xrightarrow{a}q'$ and $p'Rq'(a\in A_\tau)$.
- if pRq and $q\xrightarrow{a}q'$, then $\exists p':p\xrightarrow{a}p'$ and $p'Rq'(a\in A_\tau)$.
- if pRq then: $p\xrightarrow{a}\surd$ if and only if $q\xrightarrow{a}\surd(a\in A_\tau)$.

p and $q\in\mathcal{P}$ are *bisimilar*, notation $p\leftrightarroweq q$, if there exists a bisimulation R on $\mathcal{P}$ with pRq.

Proposition 2. *$\leftrightarroweq$ is a congruence on $\mathcal{P}$.*
Proposition 3. *Any theory for which $\mathcal{P}/\leftrightarroweq$ is a model is consistent and respects liveness.*
Proposition 4. *$\mathcal{P}/\leftrightarroweq$ is a model of* ACP$_\tau$+PR+RDP+CA+KFAR.
Proposition 5. ACP$_\tau$+PR *is a complete axiomatisation of $\mathcal{P}/\leftrightarroweq$ for closed recursion free process expressions.*
Proposition 6. *$\mathcal{P}/\leftrightarroweq$ is isomorphic to the graph model $\mathbf{G}_{\aleph_1}/\leftrightarroweq_{rd}$* of Baeten, Bergstra & Klop [2].

Proofs. Omitted.

The notion of bisimulation originates from Park [12]. Bisimilarity is similar to the notion of observation congruence of Milner [11] and rooted $\tau\delta$-bisimilarity of Baeten, Bergstra & Klop [2]. If the relations $p\xrightarrow{a}q$ and $p\xrightarrow{a}\surd$ were defined without the τ-laws of table 3, the corresponding version of bisimilarity would be strong congruence, or δ-bisimilarity; if they were defined, using an extra rule $x\xrightarrow{\tau}x$, it would be observation equivalence or $\tau\delta$-bisimilarity. In [11] and [2] observation equivalence or $\tau\delta$-bisimilarity appears as a natural equivalence, with the unpleasant property of not being a congruence. Then a context requirement or rootedness condition is proposed to make it into a congruence. This is not necessary in the present approach: bisimilarity turned out to be a congruence in a natural way.

The model $\mathcal{P}/\leftrightarroweq$ can be used to prove that the Recursive Definition Principle holds in the graph model of [2]. RDP holds trivially in $\mathcal{P}/\leftrightarroweq$: $<x\mid E>/\leftrightarroweq$ is the x-component of a solution of E in $\mathcal{P}/\leftrightarroweq$. From the last proposition it follows

that it holds in the graph model also. Details are omitted here.

As demonstrated already in BAETEN, BERGSTRA & KLOP [2], AIP does not hold in $\mathbf{G}_{\aleph_1}/\leftrightarrows_{rb}$ and hence not in $\mathcal{P}/\leftrightarrows$: Let $\sum_{n>0} a^n$ be the process $(<x \mid x=xa+a>)_{\leftrightarrows} \in \mathcal{P}/\leftrightarrows$ and $a^\omega = (<x \mid x=ax>)_{\leftrightarrows} \in \mathcal{P}/\leftrightarrows$. Then

$$\mathrm{AIP} \vdash \sum_{n>0} a^n = \sum_{n>0} a^n + a^\omega \text{ but not } \sum_{n>0} a^n \leftrightarrows \sum_{n>0} a^n + a^\omega.$$

Hence it seems worthwhile to look for another model of ACP_τ, in which AIP is valid. However, such an attempt can only succeed at the expense of RDP, CA or KFAR.

10. THE INCONSISTENCY OF BPA^* + RDP + AIP + CA + KFAR

In this section it will be proved that the combination of RDP, AIP, CA, and KFAR is inconsistent on top of ACP_τ+PR. Since the operators $\|$, $\lfloor\!\lfloor$, $|$ and ∂_H are not involved in this proof, the result can be formulated more sharply. Let BPA^* be the subtheory of ACP_τ+PR consisting of the axioms A, T, TI, and PR of table 2. Assume that the alphabet A contains at least two different actions a and b.

THEOREM 1. $\mathrm{BPA}^*+\mathrm{RDP}+\mathrm{AIP}+\mathrm{CA}+\mathrm{KFAR} \vdash \tau = \tau+\tau\delta$.

PROOF. Declare the following recursive specifications:

$$X_k = aX_{k+1}+b^k \ (k>0) \qquad Y = bY \qquad Z = aZ+\tau$$

Now the theorem follows from the following 6 lemmas:

I.	$\tau_{\{b\}}(X_1) = Z$	an application of AIP
II.	$\tau_{\{a\}}(Z) = \tau$	an application of KFAR, even of KFAR^-
III.	$\tau_{\{a,b\}}(X_1) = \tau$	from I and II, using CA
IV.	$\tau_{\{a\}}(X_1) = \tau_{\{a\}}(X_1)+Y$	an application of AIP, using T2
V.	$\tau_{\{b\}}(Y) = \tau\delta$	an application of KFAR; this time of *deadlock* = *livelock*
VI.	$\tau_{\{a,b\}}(X_1) = \tau+\tau\delta$	from IV and V, using CA and III. □

To illustrate the proof, the process graphs of X_1, $\tau_{\{b\}}(X_1)$, Z, $\tau_{\{a\}}(X_1)$ and $\tau_{\{a\}}(X_1)+Y$ are presented in figure 3. In [2,4,5,8] it can be found how process graphs can be obtained from closed process expressions.

In the proof of theorem 1, the equation of deadlock and livelock plays a crucial role. If KFAR is replaced by the weaker proof rule KFAR^-, only expressing fairness, the inconsistency disappears:

PROPOSITION 7. $\mathrm{BPA}^*+\mathrm{RDP}+\mathrm{AIP}+\mathrm{CA}+\mathrm{KFAR}^-$ *is consistent.*
PROOF. See VAN GLABBEEK [8].

However this theory has another disadvantage, it violates liveness:

PROPOSITION 8. $\mathrm{BPA}^*+\mathrm{RDP}+\mathrm{AIP}+\mathrm{CA}+\mathrm{KFAR}^- \vdash \tau = \tau+\tau^\omega$.
PROOF. As above, but without using Lemma V, it can be proved that $\tau = \tau+\tau_{\{b\}}(Y)$, where $\tau_{\{b\}}(Y)$ can be written as τ^ω (see section 8).

Furthermore in [8] it is shown that if in $\mathrm{BPA}^*+\mathrm{RDP}+\mathrm{AIP}+\mathrm{CA}+\mathrm{KFAR}$ either RDP or AIP or CA or KFAR is dropped, a consistent theory that respects liveness remains. So in theorem 1 all ingredients are really needed.

11. BOUNDED NONDETERMINISM AND THE VALIDITY OF AIP^-

Let B be a new building block that can be added to ACP_τ. It consists of the unary predicates B_n (*boundedness*) for $n \in \mathbb{N}$, and the rules and axioms presented in the fourth block of table 2. $B_n(x)$ states that the nondeterminism displayed by x before its n^{th} visible step is bounded. This means that for any sequence σ of length $< n$ of visible actions there are only finitely many different processes to which x can evolve by performing σ.

Define the predicates $\xrightarrow{\sigma}\!\!\!\!\rightarrow$ on $\mathcal{P}/\leftrightarrows$ by: $P \xrightarrow{\sigma}\!\!\!\!\rightarrow Q$ if there are $p \in P$ and $q \in Q$ with $p \xrightarrow{\sigma}\!\!\!\!\rightarrow q$. Now the predicates B_n can be defined on $\mathcal{P}/\leftrightarrows$ by: $B_n(P)$ if $\{Q \in \mathcal{P}/\leftrightarrows \mid P \xrightarrow{\sigma}\!\!\!\!\rightarrow Q\}$ is finite for any $\sigma \in A^*$ with length $<n$. $P \in \mathcal{P}/\leftrightarrows$ is *bounded* (P displays only bounded nondeterminism) if $\{Q \in \mathcal{P}/\leftrightarrows \mid P \xrightarrow{\sigma}\!\!\!\!\rightarrow Q\}$ is finite for any $\sigma \in A^*$. Of course P is bounded if and only if for all $n \in \mathbb{N}$ $B_n(P)$.

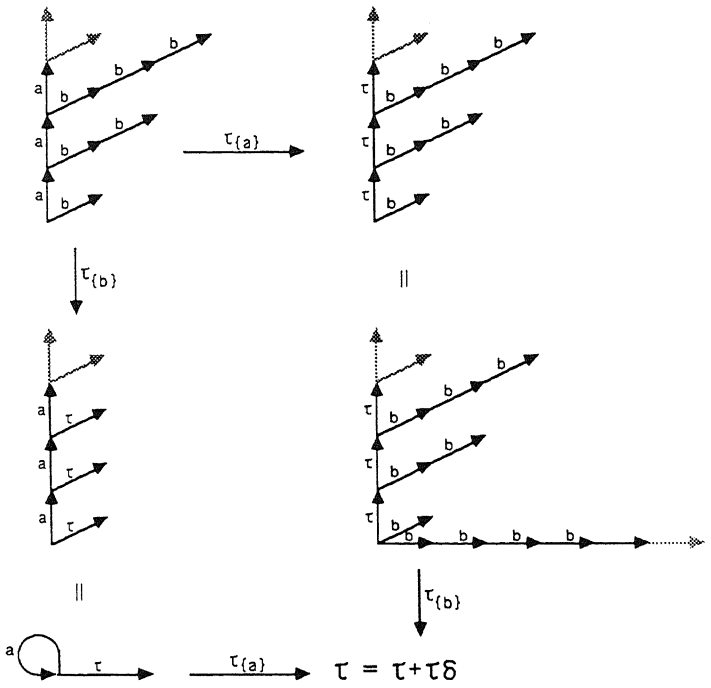

fig. 3

PROPOSITION 9. $\mathcal{P}/\leftrightarrows \models B$.
PROOF. Omitted.

A process expression $p \in \mathcal{P}$ is *(syntactically) bounded* if $\{q \in \mathcal{P} \mid p \xrightarrow{\sigma} \!\!\!\gg q\}$ is finite for any $\sigma \in A^*$. Note that $(q)\leftrightarrows \in \mathcal{P}/\leftrightarrows$ is bounded does not imply that $q \in \mathcal{P}$ is bounded.

LEMMA. *If $p \in \mathcal{P}$ is bounded and $p \xrightarrow{\sigma} \!\!\!\gg q$, then also $q \in \mathcal{P}$ is bounded.*
LEMMA. *If $P \in \mathcal{P}/\leftrightarrows$ is bounded, then there is a $p \in P$ bounded.*
PROOFS. Omitted.

Since AIP is not valid in the model $\mathcal{P}/\leftrightarrows$ and even inconsistent in combination with RDP, CA and KFAR, a restricted version AIP^- is proposed in table 2, formulated with the help of the predicates B_n. Now it remains to be proven that $\mathcal{P}/\leftrightarrows \models \text{AIP}^-$.

The proof below can be viewed as a reconstruction of the proof of BAETEN, BERGSTRA & KLOP [2], that a more restrictive version of AIP^- holds in the graph model $\mathbf{G}_{\mathbb{N}_\tau}/\leftrightarrows_{rt}$, which is isomorphic to $\mathcal{P}/\leftrightarrows$. It makes use of the lemmas above. As a corollary it follows that all rules of table 2 are satisfied by $\mathcal{P}/\leftrightarrows$, and that $\text{ACP}_\tau + \text{PR} + \text{B} + \text{RDP} + \text{AIP}^- + \text{CA} + \text{KFAR}$ is consistent and respects liveness.

THEOREM 2. $\mathcal{P}/\leftrightarrows \models \text{AIP}^-$.

PROOF. Let $P, Q \in \mathcal{P}/\leftrightarrows$, $B_n(Q)$ for $n \in \mathbb{N}$ and $\forall n \in \mathbb{N}$: $\pi_n(P) = \pi_n(Q)$. It has to be proved that $P = Q$. Take $p \in P$ and $q \in Q$, such that q is bounded. Then $\forall n \in \mathbb{N}$: $\pi_n(p) \leftrightarrows \pi_n(q)$. It suffices to prove that $p \leftrightarrows q$, i.e. that there is a bisimulation R on $\mathcal{P}$ with pRq.
CLAIM: R can be defined by: pRq if $\forall n$: $\pi_n(p) \leftrightarrows \pi_n(q)$ and q is bounded.

- Suppose pRq and $p \xrightarrow{a} p'$ (with $a \in A_\tau$). Then put $S_n = \{q^* \in \mathcal{P} \mid q \xrightarrow{a} q^* \;\&\; \pi_n(p') \leftrightarrows \pi_n(q^*)\}$, and remark that
  I. $S_0 \supseteq S_1 \supseteq S_2 \supseteq \ldots$, since $\pi_{n+1}(p') \leftrightarrows \pi_{n+1}(q^*)$ implies $\pi_n(p') \leftrightarrows \pi_n(q^*)$.
  II. $S_n \neq \varnothing$, for $n \in \mathbb{N}$, since $\pi_{n+1}(p) \leftrightarrows \pi_{n+1}(q)$.
  III. S_n is finite, for $n \in \mathbb{N}$, since q is bounded.
  From these observations it follows that $\bigcap_{n=0}^{\infty} S_n \neq \varnothing$. Choose $q' \in \bigcap_{n=0}^{\infty} S_n$, then $q \xrightarrow{a} q'$ and $p'Rq'$.
- Suppose pRq and $q \xrightarrow{a} q'$ (with $a \in A_\tau$). Then put $S_n = \{p^* \in \mathcal{P} \mid p \xrightarrow{a} p^* \;\&\; \pi_n(p^*) \leftrightarrows \pi_n(q')\}$, and remark that

$S_0 \supseteq \ldots$ and $S_n \neq \emptyset$ for $n \in \mathbb{N}$ (as above).

Now, for $n \in \mathbb{N}$, choose $p_n \in S_n$. By the first part of this proof, there are $q_n \in \mathcal{P}$ with $q \xrightarrow{a} q_n$ and $p_n R q_n$. But since q is bounded, there must be a process q^* in the sequence $q_0, q_1, q_2, \ldots$ occurring infinitely many times. Let $I = \{n \in \mathbb{N} \mid p_n R q^*\}$ and choose $i \in I$. It suffices to prove that $p_i R q'$. Let $n \in \mathbb{N}$, then an $m \in I$ exists with $m > n$. So $\pi_n(p_m) \leftrightarrows \pi_n(q')$, since $p_m \in S_m \subseteq S_n$. Furthermore $p_m R q^*$ and $p_i R q^*$, so $\pi_n(p_i) \leftrightarrows \pi_n(q^*) \leftrightarrows \pi_n(p_m) \leftrightarrows \pi_n(q')$. This holds for any $n \in \mathbb{N}$, thus $p_i R q'$.

- If pRq then: $p \xrightarrow{a} \surd \Leftrightarrow \pi_1(p) \xrightarrow{a} \surd \Leftrightarrow \pi_1(q) \xrightarrow{a} \surd \Leftrightarrow q \xrightarrow{a} \surd$.

Thus R is a bisimulation and the theorem is proved. □

12. Conclusion

In this paper a model $\mathcal{P}/\leftrightarrows$ of ACP_τ has been constructed, satisfying RDP, CA and KFAR, but not satisfying AIP. It has been shown that the price of changing this model in such a way that AIP holds is rather high:

- either RDP has to be dropped, in which case a lot of interesting processes can not be defined anymore,
- or CA has to be dropped, which makes the model very unnatural,
- or KFAR has to be dropped entirely, which makes for instance protocol verification with channels that can make errors almost impossible,
- or KFAR has to be replaced by $KFAR^-$, in which case only safety properties of protocols can be verified, and no liveness properties.

Therefore another strategy has been pursued: to find a restricted version of AIP, valid in the model $\mathcal{P}/\leftrightarrows$, whose computational possibilities approximate those of AIP as close as possible. This was first done in Baeten, Bergstra & Klop [2]. In table 2 of the present paper, a simpler and less restrictive version of AIP, called AIP^-, is proposed, not containing guarded recursive specifications as a parameter. For this reason the predicates B_n were introduced.
Finally AIP^- is proved valid in $\mathcal{P}/\leftrightarrows$, which guarantees the consistency of the resulting theory $ACP_\tau + PR + B + RDP + AIP^- + CA + KFAR$.

References

[1] J.C.M. Baeten, J.A. Bergstra & J.W. Klop, *Conditional axioms and α/β calculus in process algebra,* report CS-R8502, Centrum voor Wiskunde en Informatica, Amsterdam 1985, to appear in: Proc. IFIP Conference on Formal Description of Programming Concepts, Gl. Avernaes 1986, (M. Wirsing, ed.), North-Holland.

[2] J.C.M. Baeten, J.A. Bergstra & J.W. Klop, *On the consistency of Koomen's Fair Abstraction Rule,* report CS-R8511, Centrum voor Wiskunde en Informatica, Amsterdam 1985, to appear in Theoretical Computer Science.

[3] J.W. de Bakker & J.I. Zucker, *Processes and the denotational semantics of concurrency,* Information & Control 54 (1/2), pp. 70-120, 1982.

[4] J.A. Bergstra & J.W. Klop, *Algebra of communicating processes,* Proc. of the CWI Symp. Math. & Comp. Sci., eds. J.W. de Bakker, M. Hazewinkel & J.K. Lenstra, Amsterdam 1986.

[5] J.A. Bergstra & J.W. Klop, *Algebra of communicating processes with abstraction,* Theoretical Computer Science 37(1), pp. 77-121, 1985.

[6] J.A. Bergstra, J.W. Klop & E.-R. Olderog, *Failures without chaos: a new process semantics with fair abstraction,* report CS-R8625, Centrum voor Wiskunde en Informatica, Amsterdam 1986, to appear in: Proc. IFIP Conference on Formal Description of Programming Concepts, Gl. Avernaes 1986, (M. Wirsing, ed.), North-Holland.

[7] S.D. Brookes, C.A.R. Hoare & W. Roscoe, *A theory of communicating sequential processes,* Journal ACM 31(3), pp. 560-599, 1984.

[8] R.J. van Glabbeek, *Bounded nondeterminism and the approximation induction principle in process algebra,* report CS-R8634, Centrum voor Wiskunde en Informatica, Amsterdam 1986.

[9] G.J. Milne, *CIRCAL and the representation of communication, concurrency, and time,* Transactions on Programming Languages and Systems (ACM) 7(2), pp. 270-298, 1985.

[10] R. Milner, *A calculus for communicating systems,* Springer LNCS 92, 1980.

[11] R. Milner, *Lectures on a calculus for communicating systems, Seminar on Concurrency,* Springer LNCS 197, pp. 197-220, 1985.

[12] D.M.R. Park, *Concurrency and automata on infinite sequences,* Proc. 5th GI Conference, Springer LNCS 104, 1981.

[13] W. Reisig, *Petri Nets, An Introduction,* EATCS Monographs on Theoretical Computer Science, Springer-Verlag 1985.

[14] M. Rem, *Partially ordered computations, with applications to VLSI design,* Proc. 4th Advanced Course on Foundations of Computer Science, part 2, eds. J.W. de Bakker & J. van Leeuwen, Tract 159, Mathematisch Centrum, Amsterdam 1983.