Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

R.J. van Glabbeek

Bounded nondeterminism and the approximation induction principle in process algebra

Computer Science/Department of Software Technology Report CS-R8634 September

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

69F12, 69F32, 69F43, 69D41

Bounded Nondeterminism and the Approximation Induction Principle in Process Algebra

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. This report will be submitted for publication elsewhere.

TABLE OF CONTENTS

Report CS-R8634
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

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 Petrinets (see for instance REISIG [12]), Topological models (as in DE BAKKER & ZUCKER [3]), Algebraical models (like the projective limit models in BERGSTRA & KLOP [4]), Graph models (as in MILNER [9] 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 [13]). 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 [9]. 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 [8] 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. Only the proofs of theorems 10, 12 and 14 require knowledge of [6]. These theorems are not essential for the conclusions in section 15.

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 $|: A \times A \to A$. There the last example would be $A = \{a,b,c\}$ and $a|b = c$.

2. A Language for Communicating Processes

The language employed in this paper is built inductively from a set $V = \{x,y,z,...\}$ of variables, and the constants, functions and predicates of the signature Σ 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 Σ.

Σ:	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 $I \subseteq A$
		π_n	projection, for all $n \in \mathbb{N}$
	binary operators:	$+$	alternative composition (sum)
		$\cdot$	sequential composition (product)
		$\|\|$	parallel composition (merge)
		$\lfloor\!\lfloor$	left-merge
		$\|$	communication merge (bar)
	unary predicates:	B_n	boundedness up to level n ($n \in \mathbb{N}$)

Table 1

The meaning of these constructs will be given informally below, together with an explanation of the axioms of table 3 (on page 9).

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

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, first making a choice between its summands x and y, and then proceeding with the chosen course of action. There is no order in the presented alternatives (axiom A1) and a sequence of choices can be regarded as a single choice between all occurring alternatives (A2). Furthermore a choice between two identical alternatives is neglected (A3) and in the presence of an alternative, δ is never chosen (A6). So δ represents deadlock only if it is not occurring in a sum context.

$x \cdot y$ 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.

$x \lfloor\!\lfloor y$ 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).

$\pi_n(x)$ represents the process x, which is only allowed to perform n visible actions. The next visible action is blocked, i.e. renamed into δ (PR).

$B_n(x)$ 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 (B).

3. Action Relations between Processes

Let us enlarge the signature Σ with the binary predicates $\xrightarrow{a}$ and the unary predicates $\xrightarrow{a}\surd$, both for $a \in A_\tau$.

$x \xrightarrow{a} y$ means that the process x can evolve into y during a period in which only the action a is performed.

a :	$a \xrightarrow{a} \surd$				
$+$:	$\dfrac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'}$	$\dfrac{x \xrightarrow{a} \surd}{x+y \xrightarrow{a} \surd}$	$\dfrac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}$	$\dfrac{y \xrightarrow{a} \surd}{x+y \xrightarrow{a} \surd}$	
$\cdot$:	$\dfrac{x \xrightarrow{a} x'}{xy \xrightarrow{a} x'y}$	$\dfrac{x \xrightarrow{a} \surd}{xy \xrightarrow{a} y}$			
$\|\|$:	$\dfrac{x \xrightarrow{a} x'}{x\|\|y \xrightarrow{a} x'\|\|y}$	$\dfrac{x \xrightarrow{a} \surd}{x\|\|y \xrightarrow{a} y}$	$\dfrac{y \xrightarrow{a} y'}{x\|\|y \xrightarrow{a} x\|\|y'}$	$\dfrac{y \xrightarrow{a} \surd}{x\|\|y \xrightarrow{a} x}$	
	$\dfrac{x \xrightarrow{a} x', y \xrightarrow{b} y'}{x\|\|y \xrightarrow{a\|b} x'\|\|y'}$	$\dfrac{x \xrightarrow{a} \surd, y \xrightarrow{b} y'}{x\|\|y \xrightarrow{a\|b} y'}$	$\dfrac{x \xrightarrow{a} x', y \xrightarrow{b} \surd}{x\|\|y \xrightarrow{a\|b} x'}$	$\dfrac{x \xrightarrow{a} \surd, y \xrightarrow{b} \surd}{x\|\|y \xrightarrow{a\|b} \surd}$	(if $a\|b \in A$)
$\lfloor\!\lfloor$:	$\dfrac{x \xrightarrow{a} x'}{x \lfloor\!\lfloor y \xrightarrow{a} x\|\|y}$	$\dfrac{x \xrightarrow{a} \surd}{x \lfloor\!\lfloor y \xrightarrow{a} y}$			
$\|$:	$\dfrac{x \xrightarrow{a} x', y \xrightarrow{b} y'}{x\|y \xrightarrow{a\|b} x'\|\|y'}$	$\dfrac{x \xrightarrow{a} \surd, y \xrightarrow{b} y'}{x\|y \xrightarrow{a\|b} y'}$	$\dfrac{x \xrightarrow{a} x', y \xrightarrow{b} \surd}{x\|y \xrightarrow{a\|b} x'}$	$\dfrac{x \xrightarrow{a} \surd, y \xrightarrow{b} \surd}{x\|y \xrightarrow{a\|b} \surd}$	(if $a\|b \in A$)
∂_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:	$a \xrightarrow{a} \tau$	$\dfrac{x \xrightarrow{\tau} y,\ y \xrightarrow{a} z}{x \xrightarrow{a} z}$	$\dfrac{x \xrightarrow{\tau} y,\ y \xrightarrow{a} \surd}{x \xrightarrow{a} \surd}$	$\dfrac{x \xrightarrow{a} y,\ y \xrightarrow{\tau} z}{x \xrightarrow{a} z}$	$\dfrac{x \xrightarrow{a} y,\ y \xrightarrow{\tau} \surd}{x \xrightarrow{a} \surd}$

Table 2

$x \xrightarrow{\tau} y$ means that x can evolve into y, taking a positive amount of time in which no visible actions occur.

$x \xrightarrow{a} \surd$ means that x can terminate after having done only an a-step, and

$x \xrightarrow{\tau} \surd$ means that x can terminate after some time, without performing visible actions.

A proof system for these action relations is presented in table 2. All these rules are in fact schemes in a and b, with a,b ranging over A_τ, unless further restrictions are made in the table. The rules for recursion will be explained in section 5; they may be skipped for the moment. Let $\mathcal{P}$ denote the set of closed process expressions (= closed Σ-terms). The proof system of table 2 will be called $\overrightarrow{\text{ACP}}_\tau$. $\overrightarrow{\text{ACP}}_\tau$ is sound and complete on $\mathcal{P}$ with respect to the intended interpretation of process expressions. In this way $\overrightarrow{\text{ACP}}_\tau$ provides an alternative explanation of the meaning of the Σ-operators. Write $p \xrightarrow{a} q$, with $p,q \in \mathcal{P}$ and $a \in A_\tau$, if $\overrightarrow{\text{ACP}}_\tau \vdash p \xrightarrow{a} q$ and $p \xrightarrow{a} \surd$ if $\overrightarrow{\text{ACP}}_\tau \vdash p \xrightarrow{a} \surd$. Write $p \xrightarrow{a}$ if either $p \xrightarrow{a} \surd$ or $p \xrightarrow{a} q$ for some $q \in \mathcal{P}$.

A closed process expression may be regarded as a specification or description of a process. All information concerning where or when a process takes place is neglected, so there are a lot of processes, meeting the same specification. However, process expressions do provide some information about the timing of the actions in a process. The process a for instance has to start with an a-step immediately, while τa may wait some time first (see figure 1). The reason for making this distinction is the following. 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 $\partial_H(\cdot)$. This motivates the distinction made, between τa and a. In terms of action relations the difference can be stated as follows: $\tau a \xrightarrow{\tau} a$, but not $a \xrightarrow{\tau} a$.

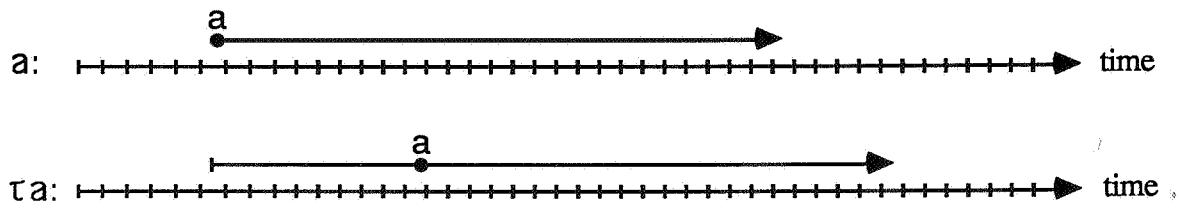

fig. 1

As suggested by figure 1, the essential features of a process, specified by a process expression a or τa are its beginning, the occurrence of the action a and its termination. In the process a, the first two coincide. The termination of a process is detectable by means of sequential composition: if b is scheduled right after a, the termination of a coincides with the beginning of b.

Using the axioms of table 3 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.

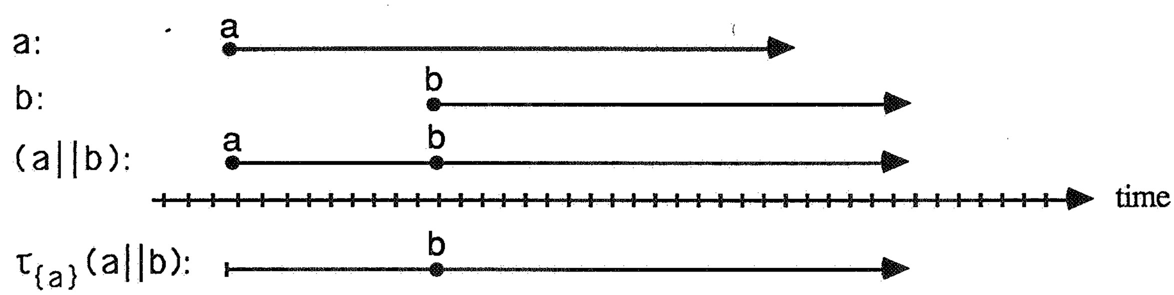

fig. 2

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.

Actually $\tau_{\{a\}}(a \| b)$ has also a summand $b\tau$, reflecting the possibility that the process b starts before a does. In the following lines it will be explained why this summand can be calculated away. The axiom $a \xrightarrow{a} \tau$ says that after the atomic action a in the process a is observed, it may still take some time before the process terminates. It reflects the assumption that, although the time interval between two atomic actions in a process may be arbitrary long, atomic actions are observed pointwise in time. Now b and $b\tau$ are identified (by the calculus presented in the next section) since their behaviour, as far as it can be expressed in terms of action relations, is the same: $b \xrightarrow{b} \surd$ and $b \xrightarrow{b} \tau$ versus $b\tau \xrightarrow{b} \surd$ and $b\tau \xrightarrow{b} \tau$ (and $b\tau \xrightarrow{b} \tau\tau$, but $\tau\tau$ and τ also behave similarly). For the same reason τb and $\tau b + b$ are identified: $\tau b \xrightarrow{b} \surd$, $\tau b \xrightarrow{b} \tau$, $\tau b \xrightarrow{\tau} b$, $\tau b \xrightarrow{\tau} \tau b$ versus $\tau b + b \xrightarrow{b} \surd$, $\tau b + b \xrightarrow{b} \tau$, $\tau b + b \xrightarrow{\tau} b$, $\tau b + b \xrightarrow{\tau} \tau b$. The second τ-law of table 2 makes clear that although the process expression $p + aq$ denotes a process that can do an a-step immediately, the action relation $p + aq \xrightarrow{a} q$ does not imply this. Thus the processes τb and $\tau b + b$ are different (only $\tau b + b$ can do an b-step immediately) but the difference cannot be expressed in terms of action relations. In the calculus of the next section, only processes are distinguished, whose difference can be expressed in terms of action relations. One might suggest to use relations $\overset{a}{\twoheadrightarrow}$ rather than $\xrightarrow{a}$, with $x \overset{a}{\twoheadrightarrow} y$ meaning that x can evolve into y, after having done an a-step immediately. However, this would lead to a semantics in which the axioms CM 2 and 3 of table 3 would not be satisfied. Therefore this option is not pursued in this paper.

Note that there is a difference between the τ-laws and the other action rules of table 2. While the other action rules describe what is really happening, the τ-laws describe also what can be observed, considering the invisibility of the process τ. Therefore the part of $\overrightarrow{ACP}_\tau$ without the τ-laws can be called concrete $\overrightarrow{ACP}_\tau$, while the entire proof system can be called abstract.

Action relations can be regarded as a way to obtain a process graph (= labeled transition diagram) for each closed process expression (in the obvious way). The edges of such a graph are labeled by elements of A_τ and the end nodes may be labeled by $\surd$. The $\surd$-label represents termination, its absence (in an end node) represents deadlock. Following the distinction made above, concrete and abstract graphical representations of process expressions can be distinguished. An example should suffice to explain the idea. The concrete process graph of $a(\tau b + c\delta)$ is shown in figure 3a, while its abstract version can be found in figure 3c. Figure 3b gives an intermediate version in which only the first τ-law is not used.

a)

b)

c)

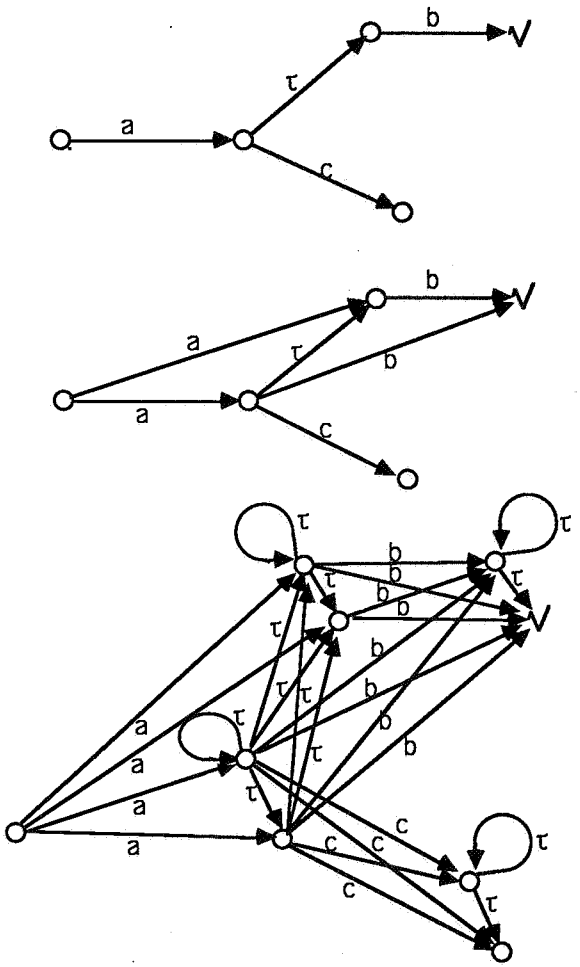

fig. 3

The names of the nodes in 3*a*,*b* are $a(\tau b + c\delta)$, $\tau b + c\delta$, b, $\surd$, and δ. In 3c also the nodes $\tau(\tau b + c\delta)$, τb, τ and $\tau\delta$ occur. In general the intermediate version can be obtained from the concrete one, by completing triangles; while in the abstract version each node (= state = subprocess) is split into two processes: an active one that has to do some action (or to lose some option) immediately, and a passive one that may wait some time first. The *process graph* of $p \in \mathcal{P}$ will denote the concrete version.

Notice that in the abstract process graph τ-loops occur. They originate from the axiom $\tau \xrightarrow{\tau} \tau$, which is an instance of the first τ-law. $\tau \xrightarrow{\tau} \tau$ means that after waiting some time on a process to terminate (or in general to reach another state) it may still not be ready. However, this implies by no means that the process may fail to terminate.

4. 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 3. As in table 2, all axioms are in fact axiom schemes in a,b and c, but this time a,b,c range over A_δ, unless further restrictions are made in the table. 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

ACP_τ	$x+y=y+x$	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		
	$a\mid b=b\mid a$	C1		
	$(a\mid b)\mid c=a\mid(b\mid c)$	C2		
	$a\mid b=\delta$ if $a\mid b\notin A$	C3		
	$x\Vert y=x\lfloor\!\lfloor y+y\lfloor\!\lfloor x+x\mid y$	CM1		
	$a\lfloor\!\lfloor x=ax$	CM2	$\tau\lfloor\!\lfloor x=\tau x$	TM1
	$(ax)\lfloor\!\lfloor y=a(x\Vert y)$	CM3	$(\tau x)\lfloor\!\lfloor y=\tau(x\Vert y)$	TM2
	$(x+y)\lfloor\!\lfloor z=x\lfloor\!\lfloor z+y\lfloor\!\lfloor z$	CM4	$\tau\mid x=\delta$	TC1
	$(ax)\mid b=(a\mid b)x$	CM5	$x\mid\tau=\delta$	TC2
	$a\mid(bx)=(a\mid b)x$	CM6	$(\tau x)\mid y=x\mid y$	TC3
	$(ax)\mid(by)=(a\mid b)(x\Vert y)$	CM7	$x\mid(\tau y)=x\mid y$	TC4
	$(x+y)\mid z=x\mid z+y\mid z$	CM8		
	$x\mid(y+z)=x\mid y+x\mid z$	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	$B_0(x)$ $B_n(a)$ $B_n(\tau)$ $\dfrac{B_n(x)}{B_n(\tau x)}$ $\dfrac{B_n(x)}{B_{n+1}(ax)}$ $\dfrac{B_n(x),B_n(y)}{B_n(x+y)}$			
AIP^-	$\dfrac{\forall n\in\mathbb{N}\ \pi_n(x)=\pi_n(y),B_n(x)}{x=y}$			
KFAR	$\dfrac{x=ix+y}{\tau_{\{i\}}(x)=\tau\cdot\tau_{\{i\}}(y)}$			
CA	$\tau_I\circ\tau_J(x)=\tau_{I\cup J}(x)$			

Table 3

from MILNER [9]. They reflect the same observational abstractions as the τ-laws in table 2; two different processes are identified if they are in a certain sense indistinguishable. Apparently, τ-steps do not engage in communications (TC1,2) but in combination with the second τ-law (of table 2 and 3) this leads to an equation $\tau x \mid y = x \mid y$, rather then $\tau x \mid y = \delta$ (TC 3,4). All other axioms are explained already in section 2. ACP_τ is a theory over the signature $\Sigma - \{\pi_n, B_n \mid n \in \mathbb{N}\}$. Axioms for the full signature Σ are added in the blocks PR (projection) and B (boundedness). As usual, these axioms (and rules) form a calculus, together with the congruence rules, which are mostly not stated explicitly. However, for this occasion, they can be found in table 4.

	$x = x$	$\dfrac{x=y}{y=x}$	$\dfrac{x=y,\ y=z}{x=z}$	
$\dfrac{x=x', y=y'}{x+y=x'+y'}$	$\dfrac{x=x', y=y'}{xy=x'y'}$	$\dfrac{x=x', y=y'}{x \Vert y = x' \Vert y'}$	$\dfrac{x=x', y=y'}{x \between y = x' \between y'}$	$\dfrac{x=x', y=y'}{x \mid y = x' \mid y'}$
$\dfrac{x=x'}{\partial_H(x)=\partial_H(x')}$	$\dfrac{x=x'}{\tau_I(x)=\tau_I(x')}$	$\dfrac{x=x'}{\pi_n(x)=\pi_n(x')}$		$\dfrac{x=x', B_n(x)}{B_n(x')}$

Table 4

$ACP_\tau + PR + B$ is a sound and complete proof system for finite closed process expressions, with respect to the semantical notion of bisimulation. For ACP_τ this is proved in BERGSTRA & KLOP [5]. 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 as a process expression, built up inductively, following the scheme τ, ax, τx, $x + y$. The axioms CM2,5 and 6 are derivable from the others.

The rules AIP^- and KFAR, mentioned in table 3, will be explained later. The last axiom of table 4, 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.

5. 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 a process expression 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 the next section 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 $<x \mid E>$ (with $x \in V_E$) denote the x-component of this solution. If E has more than one solution, $<x \mid E>$ 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 $<x \mid E>$. In a recursive language, the syntactical constructs $<x \mid E>$ 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 justifies the notation $T + RDP \vdash e$.

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

PROOF. Omitted.

The set $\mathcal{P}$ of closed process expressions over Σ, used in section 3, is understood to contain all closed recursive process expressions. Recursion free process expressions are called *finite*. The rules for recursion in table 2, state that a process $\langle x \mid E \rangle$ behaves exactly as $\langle t_x \mid E \rangle$. They are schemes, ranging over all recursive specifications $E = \{x = t_x \mid x \in V_E\}$ and all $x \in V_E$. Now $\overrightarrow{\mathrm{ACP}}_\tau$ without the rules for recursion is sound and complete on the domain of finite closed process expressions (with respect to their intended interpretation), and $\overrightarrow{\mathrm{ACP}}_\tau$ with these rules on all of $\mathcal{P}$. Note that while $\langle x \mid E \rangle \xrightarrow{a}$ implies that *any* x-component of a solution of E can do an a-step, not $\langle x \mid E \rangle \xrightarrow{a}$ does not exclude that *some* x-component of a solution of E can do an a-step.

6. A TERM MODEL FOR ACP_τ + RDP

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 $p \leftrightarrows q$, if there exists a bisimulation R on $\mathcal{P}$ with pRq.

THEOREM 2. $\leftrightarrows$ *is a congruence on* $\mathcal{P}$.

THEOREM 3. $\mathcal{P}/\leftrightarrows$ *is a model of* $\mathrm{ACP}_\tau + \mathrm{PR} + \mathrm{RDP} + \mathrm{CA}$.

THEOREM 4. *$\mathcal{P}/\leftrightarrows$ is isomorphic to the graph model $\mathbb{G}_{\aleph_1}/\leftrightarrows_{rb}$* of BAETEN, BERGSTRA & KLOP [2].

PROOFS. Omitted.

The notion of bisimulation originates from PARK [11]. Bisimilarity is similar to the notion of observation congruence of MILNER [10] and rooted $\tau\delta$-bisimilarity of BAETEN, BERGSTRA & KLOP [2]. If the relations $p \xrightarrow{a} q$ and $p \xrightarrow{a} \surd$ were defined, using $\overrightarrow{\text{ACP}}_\tau$ without the τ-laws, the corresponding version of bisimilarity would be strong congruence, or δ-bisimilarity; if they were defined, using $\overrightarrow{\text{ACP}}_\tau \cup \{x \xrightarrow{\tau} x\}$, it would be observation equivalence or $\tau\delta$-bisimilarity. In [10] 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 $\mathcal{P}/\leftrightarrows$ can be used to prove that the Recursive Definition Principle holds in the graph model of [2]. RDP holds trivially in $\mathcal{P}/\leftrightarrows$: $<x\,|\,E>/\leftrightarrows$ is the x-component of a solution of E in $\mathcal{P}/\leftrightarrows$. From theorem 4 it follows that it holds in the graph model also. Details are omitted here.

Remark that the meaning of $<x\,|\,E> \in \mathcal{P}$ after interpretation in the model $\mathcal{P}/\leftrightarrows$ is different from the meaning of $<x\,|\,E>/\leftrightarrows \in \mathcal{P}/\leftrightarrows$. While the former denotes a class of processes, each of which is the x-component of a solution of E, the latter denotes a special element from this class, namely the one which can only do those moves $\xrightarrow{a}$, whose occurrence is provable from $\overrightarrow{\text{ACP}}_\tau$.

7. BOUNDED NONDETERMINISM

Let us enlarge the signature Σ again; this time with the binary predicates $\xrightarrow{\sigma}\!\!\!\!\!\rightarrow$ and the unary predicates $\xrightarrow{\sigma}\!\!\!\!\!\rightarrow \surd$, both for $\sigma \in A^*$. Let $\tau \in A^*$ denote the empty string. $x \xrightarrow{\sigma}\!\!\!\!\!\rightarrow y$ means that x can evolve into y during a period >0 in which (only) the sequence of actions σ is performed. $x \xrightarrow{\sigma}\!\!\!\!\!\rightarrow \surd$ means that x can terminate after a period >0 in which (only) the sequence of actions σ is performed. Let $\overrightarrow{\overrightarrow{\text{ACP}}}_\tau$ denote $\overrightarrow{\text{ACP}}_\tau$ + the rules of table 5.

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

Table 5

Here a ranges over A and σ,ρ over A^*. Write $p \xrightarrow{\sigma}\!\!\!\!\!\rightarrow q$, with $p,q \in \mathcal{P}$ and $\sigma \in A^*$, if $\overrightarrow{\overrightarrow{\text{ACP}}}_\tau \vdash p \xrightarrow{\sigma}\!\!\!\!\!\rightarrow q$; $p \xrightarrow{\sigma}\!\!\!\!\!\rightarrow \surd$ if $\overrightarrow{\overrightarrow{\text{ACP}}}_\tau \vdash p \xrightarrow{\sigma}\!\!\!\!\!\rightarrow \surd$; and $p \xrightarrow{\sigma}\!\!\!\!\!\rightarrow$ if either $p \xrightarrow{\sigma}\!\!\!\!\!\rightarrow \surd$ or $p \xrightarrow{\sigma}\!\!\!\!\!\rightarrow q$ for some $q \in \mathcal{P}$.

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 $P \in \mathcal{P}/\leftrightarrows_{rb}$ is *bounded* (P displays only bounded nondeterminism) if $\{Q \in \mathcal{P}/\leftrightarrows \mid P \xrightarrow{\sigma}\!\!\!\!\!\rightarrow Q\}$ is finite for

any $\sigma \in A^*$. The predicates B_n are defined on $\mathcal{P}/\leftrightarrows$ by: $B_n(P)$ if $\{Q \in \mathcal{P}/\leftrightarrows \mid P \xrightarrow{\sigma} Q\}$ is finite for any $\sigma \in A^*$ with length $<n$. Of course P is bounded if and only if for all $n \in \mathbb{N}$ $B_n(P)$.

THEOREM 5. $\mathcal{P}/\leftrightarrows \models B$.

THEOREM 6. $ACP_\tau + PR + B$ *is a complete axiomatisation of* $\mathcal{P}/\leftrightarrows$ *for finite closed process expressions.*

PROOFS. Omitted. In BERGSTRA & KLOP [5], ACP_τ is proved to be a complete axiomatisation for finite closed process expressions, of a graph model.

For future reference a few lemmas will be proved. A process expression $p \in \mathcal{P}$ is *(syntactically) bounded* if $\{q \in \mathcal{P} \mid p \xrightarrow{\sigma} q\}$ 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* $p \xrightarrow{\sigma} q$, *then also* $q \in \mathcal{P}$ *is bounded.*

PROOF. Trivial.

LEMMA. *If* $P \in \mathcal{P}/\leftrightarrows$ *is bounded, then there is a* $p \in P$ *bounded.*

PROOF. Associate a variable X_Q to each subprocess Q of P. (Q is a subprocess of P if $Q = P$ or $P \xrightarrow{\sigma} Q$ for some $\sigma \in A^*$). Write $X_Q = \sum_i a_i X_{Q_i} + \sum_j b_j$ (for all variables X_Q) where the first sum is taken over all pairs $a_i X_{Q_i}$ for which $Q \xrightarrow{a_i} Q_i$ and the second over all b_j for which $Q \xrightarrow{b_j} \surd$. Together these equations form a recursive specification E and $\langle X_P \mid E \rangle$ is a bounded process expression with $\langle X_P \mid E \rangle \in P$. That $\langle X_P \mid E \rangle$ is bounded can be concluded, since P is bounded and $\langle X_P \mid E \rangle \xrightarrow{\sigma} q$ implies $q = \langle X_Q \mid E \rangle$ for some $Q \in \mathcal{P}/\leftrightarrows$ with $P \xrightarrow{\sigma} Q$ or $q = \tau\langle X_Q \mid E \rangle$ for some Q with $P \xrightarrow{\sigma} Q$ or $q = \tau$. □

8. 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 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 will be discussed in the next section.

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. The model $\mathcal{P}/\leftrightarrows$ satisfies KFAR$^-$.

9. 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:

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

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:

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

The rule *deadlock = livelock* can be obtained from KFAR by substituting $y = \delta$. KFAR$^-$ can be obtained by substituting $y = \tau y + z$. In BAETEN, BERGSTRA & KLOP [2], KFAR is shown valid in the graph model $\mathbb{G}_{\aleph_1}/\leftrightarrows_{rsb}$. Therefore it also holds in the isomorphic model $\mathcal{P}/\leftrightarrows$.

10. CONSISTENCY, SAFETY & LIVENESS

Write $p \not\rightarrow$ if $p \xrightarrow{a}$ for no $a \in A_\tau$. Let $\sigma = <a_0 a_1 a_2 \ldots> \in A^\omega$ be an infinite string of actions $a_i \in A$.
Write $p \xrightarrow{\sigma\gg}$ if there are $p_i \in \mathcal{P}$ $(i \in \mathbb{N})$ such that $\overrightarrow{\text{ACP}}_\tau \vdash p \xrightarrow{a_0} p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots$.
p is *divergent*, notation $p\uparrow$, if there are $p_i \in \mathcal{P}$ $(i \in \mathbb{N})$ such that concrete $\overrightarrow{\text{ACP}}_\tau \vdash p \xrightarrow{\tau} p_0 \xrightarrow{\tau} p_1 \xrightarrow{\tau} \cdots$.
Now the following sets of traces of a closed process expression $p \in \mathcal{P}$ can be defined:

- its unfinished traces: $u\text{-}tr(p) = \{\sigma \in A^* \mid p \xrightarrow{\sigma\gg}\}$
- its termination traces: $\surd\text{-}tr(p) = \{\sigma\surd \mid p \xrightarrow{\sigma\gg} \surd\}$
- its deadlock traces: $\delta\text{-}tr(p) = \{\sigma\delta \mid \exists q \in \mathcal{P}: p \xrightarrow{\sigma\gg} q \not\rightarrow\}$
- its finite traces: $Tr(p) = u\text{-}tr(p) \cup \surd\text{-}tr(p)$
- its finite traces with δ: $Tr^\delta(p) = u\text{-}tr(p) \cup \surd\text{-}tr(p) \cup \delta\text{-}tr(p)$
- its infinite traces: $\omega\text{-}tr(p) = \{\sigma \in A^\omega \mid p \xrightarrow{\sigma\gg}\}$
- its divergence traces: $\uparrow\text{-}tr(p) = \{\sigma \in A^* \mid \exists q \in \mathcal{P}: p \xrightarrow{\sigma\gg} q \;\&\; q\uparrow\}$
- its livelock traces: $\lambda\text{-}tr(p) = \{\sigma \in A^* \mid \exists q \in \mathcal{P}: p \xrightarrow{\sigma\gg} q \;\&\; Tr^\delta(q) = \{\tau\}\}$
- its deadlock/livelock traces: $\delta\lambda\text{-}tr(p) = \{\sigma \in A^* \mid \exists q \in \mathcal{P}: p \xrightarrow{\sigma\gg} q \;\&\; Tr(q) \subseteq \{\tau\}\}$
- its complete traces with δ: $Tr^{c\delta}(p) = \omega\text{-}tr(p) \cup \surd\text{-}tr(p) \cup \delta\text{-}tr(p) \cup \uparrow\text{-}tr(p)$
- its fair traces with δ: $Tr^{f\delta}(p) = \omega\text{-}tr(p) \cup \surd\text{-}tr(p) \cup \delta\text{-}tr(p) \cup \lambda\text{-}tr(p)$
- its fair traces: $Tr^f(p) = \omega\text{-}tr(p) \cup \surd\text{-}tr(p) \cup \delta\lambda\text{-}tr(p)$

Notice that $\delta\lambda\text{-}tr(p)\delta = \delta\text{-}tr(p) \cup \lambda\text{-}tr(p)\delta$ and that $\lambda\text{-}tr(p) \subseteq \uparrow\text{-}tr(p)$.

The finite traces of p (with δ, if deadlock is considered observable, see section 9) are exactly those sequences of actions, which can be recorded in a finite time, during a run of the process. Its fair traces (with δ) are exactly those sequences which can be recorded in an infinite time (assuming fairness). A theory T is said to be:

- *trace consistent* if $T \vdash p = q \Rightarrow Tr(p) = Tr(q)$
- *δt-consistent* if $T \vdash p = q \Rightarrow Tr^{\delta}(p) = Tr^{\delta}(q)$
- *$c\delta t$-consistent* if $T \vdash p = q \Rightarrow Tr^{c\delta}(p) = Tr^{c\delta}(q)$.
- *$f\delta t$-consistent* if $T \vdash p = q \Rightarrow Tr^{f\delta}(p) = Tr^{f\delta}(q)$
- *ft-consistent* if $T \vdash p = q \Rightarrow Tr^{f}(p) = Tr^{f}(q)$

Only $c\delta t$-consistency is violated by the use of fair abstraction. Equating deadlock and livelock violates δt- $c\delta t$- and $f\delta t$-consistency. Deadlock behaviour (possibly together with livelock behaviour) is preserved by any theory which is either δt-consistent or *ft*-consistent. On the domain of finite closed process expressions δt-consistency and *ft*-consistency coincide (since livelock cannot be expressed by finite process expressions). Therefore a theory is called *consistent* if it is δt-consistent on the domain of finite closed process expressions. Thus a theory T is consistent iff $T \vdash p = q$ implies:

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

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

for any pair of finite closed 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. In this paper process theories are required to be consistent. In figure 4 the implications between the various notions of consistency are displayed.

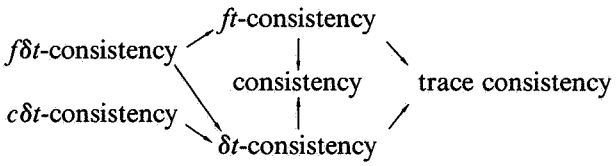

fig. 4

As remarked earlier, a process expression $p \in \mathcal{P}$ can be regarded as a specification of a process. Another process expression $q \in \mathcal{P}$ meets this specification if p and q have similar properties in some sense. A useful theory T should have the property that $T \vdash p = q$ implies that q meets the specification p. In order to make this more precise, 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. In this paper these notions will be formalised as follows:

Let $B \subseteq A^* \cup A^*\surd \cup A^*\delta$ be an arbitrary set of unfinished traces, termination traces and deadlock traces, representing the bad. Then *B can happen* to $p \in \mathcal{P}$ if $Tr^{\delta}(p) \cap B \neq \emptyset$. *$B$ can in no way happen* to $p \in \mathcal{P}$ if $Tr^{\delta}(p) \cap B = \emptyset$ and there is no $\sigma\delta \in B$ with $\sigma \in \lambda\text{-}tr(p)$. If $T \vdash p = q$ and there is a B which can happen to q, but in no way to p, then T is said to *violate safety*. Note that safety is implied by δt-consistency and *ft*-consistency and implies consistency and trace consistency.

Let $G \subseteq A^* \cup A^*\surd$ be an arbitrary set of unfinished traces and termination traces, representing the good. Then *G will eventually happen* to $p \in \mathcal{P}$ if any complete trace of p has an element of G as initial part. *G will eventually happen* to $p \in \mathcal{P}$, *assuming fairness*, if any fair trace of p has an element of G as initial part. If $T \vdash p = q$ and there is a G which will eventually happen to p, but does not have to happen to q, not even if fairness is assumed, then T is said to *violate liveness*. Note that liveness is implied by $c\delta t$-consistency and *ft*-consistency.

EXAMPLE. If $T \vdash \tau = \tau + \tau^{\omega}$, where τ^{ω} is an abbreviation of $\tau_{\{i\}}(<x \mid x = ix>)$, then T can be safe, but it violates liveness: at the right hand side the good thing $\surd$ does not have to happen eventually.

11. The Approximation Induction Principle

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. As demonstrated already in Baeten, Bergstra & Klop [2], AIP does not hold in the graph model $\mathcal{G}_{\mathbb{N}_1}/\leftrightarrows_{rrb}$ which is isomorphic to the term model $\mathcal{P}/\leftrightarrows$ of section 6:
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 AIP $\vdash \sum_{n>0} a^n = \sum_{n>0} a^n + a^\omega$ but not $\sum_{n>0} a^n \leftrightarrows \sum_{n>0} a^n + a^\omega$ (see figure 5).

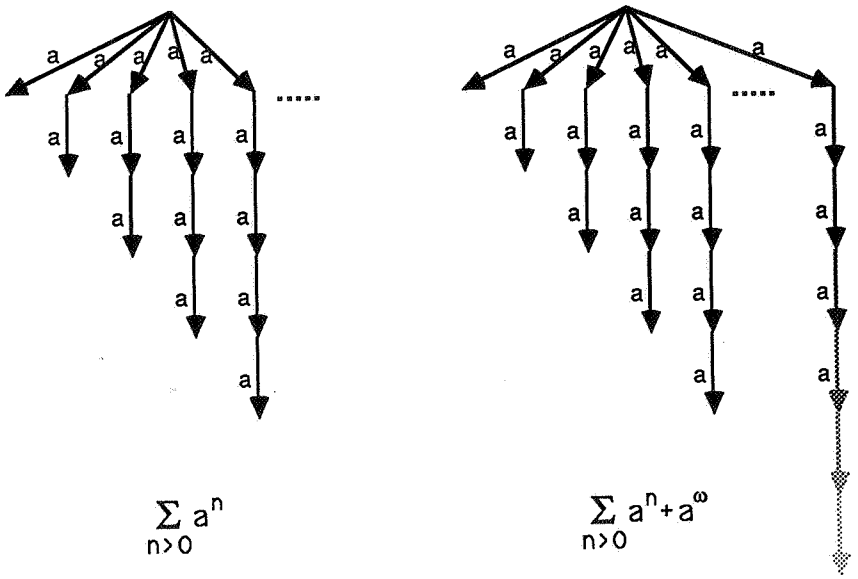

fig. 5

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.

12. 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 $ACP_\tau + PR$. 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 $ACP_\tau + PR$ consisting of the axioms A, T, TI, and PR of table 3. Assume that the alphabet A contains at least two different actions a and b.

Theorem 7. $BPA^* + RDP + AIP + CA + KFAR \vdash \tau = \tau + \tau\delta$.

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. $\tau_{\{b\}}(X_1) = Z$
II. $\tau_{\{a\}}(Z) = \tau$
III. $\tau_{\{a,b\}}(X_1) = \tau$
IV. $\tau_{\{a\}}(X_1) = \tau_{\{a\}}(X_1) + Y$
V. $\tau_{\{b\}}(Y) = \tau\delta$
VI. $\tau_{\{a,b\}}(X_1) = \tau + \tau\delta$

adI. $\tau_{\{b\}}(X_k) = a\tau_{\{b\}}(X_{k+1}) + \tau$ (for $k>0$). With induction on n it can be proved that $\pi_n(\tau_{\{b\}}(X_k)) = \pi_n(Z)$ (for $n \in \mathbb{N}$ and $k>0$):

Induction Basis: $\pi_0(\tau_{\{b\}}(X_k)) = \pi_0(a\tau_{\{b\}}(X_{k+1}) + \tau) = \delta + \tau = \pi_0(aZ + \tau) = \pi_0(Z)$.

Induction Hypothesis: $\pi_n(\tau_{\{b\}}(X_k)) = \pi_n(Z)$

Induction Step: $\pi_{n+1}(\tau_{\{b\}}(X_k)) =$

$$\pi_{n+1}(a\tau_{\{b\}}(X_{k+1}) + \tau) =$$
$$a\pi_n(\tau_{\{b\}}(X_{k+1})) + \tau = \text{(by induction hypothesis)}$$
$$a\pi_n(Z) + \tau = \pi_n(aZ + \tau) = \pi_n(Z).$$

Thus $\tau_{\{b\}}(X_k) = Z$, using AIP.

adII. This is an instance of KFAR; even of KFAR$^-$.

adIII. $\tau_{\{a,b\}}(X_1) = \tau_{\{a\}} \circ \tau_{\{b\}}(X_1) = \tau_{\{a\}}(Z) = \tau$, using CA.

adIV. This follows from AIP, since it can be proved that $\pi_n(\tau_{\{a\}}(X_k)) = \pi_n(\tau_{\{a\}}(X_k) + Y)$ for $n \in \mathbb{N}$ and $k>0$.

Case 1. $k > n$. $\pi_n \circ \tau_{\{a\}}(X_k) = \pi_n \circ \tau_{\{a\}}(aX_{k+1} + b^k) =$

$$\pi_n \circ \tau_{\{a\}}(aX_{k+1} + b^k + b^k) = \pi_n \circ \tau_{\{a\}}(X_k + b^k) =$$
$$\pi_n \circ \tau_{\{a\}}(X_k) + b^n\delta = \pi_n(\tau_{\{a\}}(X_k) + Y).$$

Case 2. $k \leqslant n$. Use induction on $n-k$:

$$\pi_n \circ \tau_{\{a\}}(X_k) = \pi_n \circ \tau_{\{a\}}(aX_{k+1} + b^k) =$$
$$\tau \cdot \pi_n \circ \tau_{\{a\}}(X_{k+1}) + b^k = \text{(by induction hypothesis)}$$
$$\tau[\pi_n(\tau_{\{a\}}(X_{k+1}) + Y)] + b^k = \text{(using T2)}$$
$$\tau[\pi_n \circ \tau_{\{a\}}(X_{k+1}) + b^n\delta] + b^k + b^n\delta = \text{(same way back)}$$
$$\pi_n \circ \tau_{\{a\}}(X_k) + b^n\delta = \pi_n(\tau_{\{a\}}(X_k) + Y).$$

adV. V is an instance of KFAR; this time of *deadlock = livelock.*

adVI. $\tau_{\{a,b\}}(X_1) = \tau_{\{b\}} \circ \tau_{\{a\}}(X_1) = \tau_{\{b\}}(\tau_{\{a\}}(X_1) + Y) = \tau_{\{a,b\}}(X_1) + \tau_{\{b\}}(Y) = \tau + \tau\delta$, 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 6.

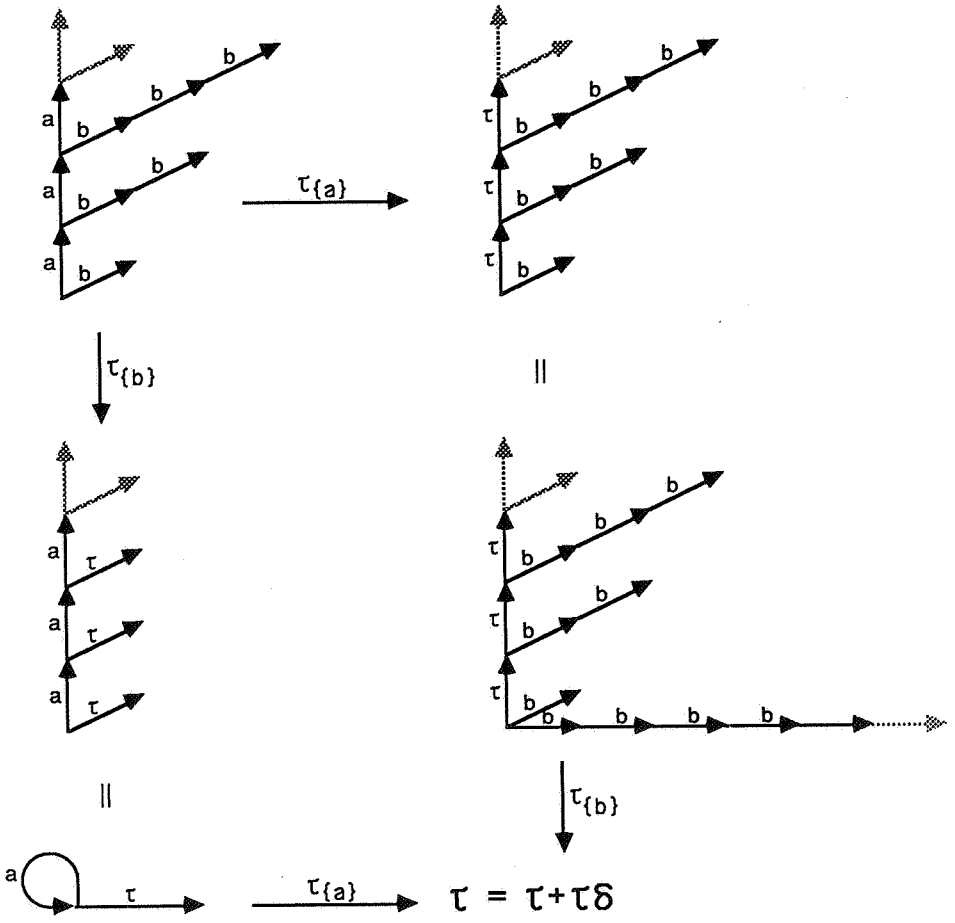

fig. 6

13. All Ingredients of this Inconsistency are really needed

A model M is said to be

- *ft*-consistent if $M \models p=q \Rightarrow Tr^f(p) = Tr^f(q)$ for $p,q \in \mathcal{P}$
- δt-consistent if $M \models p=q \Rightarrow Tr^\delta(p) = Tr^\delta(q)$ for $p,q \in \mathcal{P}$
- consistent if $M \models p=q \Rightarrow Tr^\delta(p) = Tr^\delta(q)$ for $p,q \in \mathcal{P}$ finite.

Remember that for finite $p \in \mathcal{P}$, $Tr^\delta(p)$ and $Tr^c(p)$ provide the same information. Trivially the following holds:

- M is *ft*-consistent $\Rightarrow$ M is consistent
- M is δt-consistent $\Rightarrow$ M is consistent
- M is consistent and $M \models T \Rightarrow T$ is consistent.

Now the consistency of a theory can be proved by giving a model which is either *ft*-consistent or δt-consistent.

THEOREM 8. $BPA^* + RDP + CA + KFAR$ *is consistent.*

PROOF. From theorem 3 in section 6 and the last line of section 9 it follows that:

$$\mathcal{P}/\leftrightarrows \models BPA^* + RDP + CA + KFAR$$

>From the definition of bisimulation it follows trivially that $\mathcal{P}/\leftrightarrows$ is *ft*-consistent. □

THEOREM 9. $BPA^* + AIP + CA + KFAR$ *is consistent.*

PROOF. $Q \in \mathcal{P}/\leftrightarrows$ is a *subprocess* of $P \in \mathcal{P}/\leftrightarrows$ if $P \xrightarrow{\sigma} Q$ for some $\sigma \in A^*$. $P \in \mathcal{P}/\leftrightarrows$ is *regular* if it has only finitely many subprocesses. The domain $\mathcal{R}/\leftrightarrows \subseteq \mathcal{P}/\leftrightarrows$ of regular processes is closed with respect to the operators of Σ. Furthermore all its elements are bounded. Thus from theorem 15 of section 15 it follows that:

$$\mathcal{R}/\leftrightarrows \models BPA^* + AIP + CA + KFAR.$$

>As for $\mathcal{P}/\leftrightarrows$, it follows from the definition of bisimulation that $\mathcal{R}/\leftrightarrows$ is *ft*-consistent. □

THEOREM 10. $BPA^* + RDP + AIP + CA$ *is consistent.*

PROOF. In BERGSTRA, KLOP & OLDEROG [6] a model $\mathbb{G}_{\aleph_0}/\equiv_\Delta$ of BPA^* is constructed, clearly satisfying AIP and CA. Although this model uses a domain of finitely branching process graphs, it also satisfies RDP. A proof of this fact is outside the scope of this paper, but a sketch of a proof follows.

Any recursive specification has a solution in $\mathcal{P}/\leftrightarrows$, which translates easily into a process graph (see section 6). Wherever this graph is infinitely branching, this originates from *unguarded recursion* (this is for insiders; a definition of guardedness is left out). Now the addition of an action rule $<x \mid E> \xrightarrow{\tau} <x \mid E>$ for any recursive specification E with x unguarded in E, leads to a similar graph, which has a τ-loop at all its infinitely branching nodes. These τ-loops can be replaced by infinite τ-paths, in such a way that the graph becomes finitely branching. The corresponding process is only the x-component of a solution of E in failure semantics, as employed in $\mathbb{G}_{\aleph_0}/\equiv_\Delta$.

$$\mathbb{G}_{\aleph_0}/\equiv_\Delta \models BPA^* + RDP + AIP + CA$$

>Note that the various notions of consistency, safety and liveness are defined in terms of action relations. Therefore they depend on the chosen set of action rules. From its definition it follows that $\mathbb{G}_{\aleph_0}/\equiv_\Delta$ is δt-consistent w.r.t. the extended set of action rules. For finite closed process expressions this is the same as δt-consistent w.r.t. $\overrightarrow{ACP}_\tau$. Hence $\mathbb{G}_{\aleph_0}/\equiv_\Delta$ is consistent. □

THEOREM 11. $BPA^* + RDP + AIP + KFAR$ *is consistent.*

PROOF. Two closed process expressions p and $q \in \mathcal{P}$ are finitely bisimilar, notation $p \leftrightarrows_\omega q$, if $\pi_n(p) \leftrightarrows \pi_n(q)$ for all $n \in \mathbb{N}$. $\leftrightarrows_\omega$ is a congruence on $\mathcal{P}$ for all operators of Σ, except for τ_I. Now define the *closed action relations* $\xrightarrow{a}_{cl}$ and $\xrightarrow{a}_{cl}\surd$ on $\mathcal{P}$ as the relations generated by $\overrightarrow{ACP}_\tau$ and the rules of table 6.

$$\frac{p \leftrightarrows_\omega q,\ p \xrightarrow{a} r}{q \xrightarrow{a} r} \qquad \frac{p \leftrightarrows_\omega q,\ p \xrightarrow{a} \surd}{q \xrightarrow{a} \surd}$$

Table 6

Here a ranges over A_τ and p,q,r over $\mathcal{P}$. With the help of these relations *closed bisimilarity* $\leftrightarrows_{cl} \subseteq \mathcal{P} \times \mathcal{P}$ can be defined as $\leftrightarrows$ in section 6.

CONJECTURE: $\leftrightarrows_{cl}$ is a congruence on $\mathcal{P}$ and $\mathcal{P}/\leftrightarrows_{cl} \models \mathrm{BPA}^* + \mathrm{RDP} + \mathrm{AIP} + \mathrm{KFAR}$.

>As before $\mathcal{P}/\leftrightarrows_{cl}$ is *ft*-consistent, but this time w.r.t. $\rightarrow_{cl}$. For finite closed process expressions this is the same as *ft*-consistent w.r.t. $\overrightarrow{\mathrm{ACP}}_\tau$. Hence $\mathcal{P}/\leftrightarrows_{cl}$ is consistent. However, CA is not valid in this model, since, using the process X_1 of section 12, $\tau_{\{a\}} \circ \tau_{\{b\}}(X_1) = \tau$, while $\tau_{\{b\}} \circ \tau_{\{a\}}(X_1) = \tau(\tau + \tau\delta)$. □

14. BPA* + RDP + AIP + CA + KFAR⁻ VIOLATES LIVENESS

In the proof of theorem 7, 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:

THEOREM 12. $\mathrm{BPA}^* + \mathrm{RDP} + \mathrm{AIP} + \mathrm{CA} + \mathrm{KFAR}^-$ *is consistent.*

PROOF. The model $\mathbb{G}_{\aleph_0}/\equiv^{\cdot}$ of BERGSTRA, KLOP & OLDEROG [6] satisfies KFAR^-.

$$\mathbb{G}_{\aleph_0}/\equiv^{\cdot} \models \mathrm{BPA}^* + \mathrm{RDP} + \mathrm{AIP} + \mathrm{CA} + \mathrm{KFAR}^-$$

>As for $\mathbb{G}_{\aleph_0}/\equiv_\Delta$ it follows that $\mathbb{G}_{\aleph_0}/\equiv^{\cdot}$ is consistent. □

However this theory has another disadvantage, it violates liveness:

THEOREM 13. $\mathrm{BPA}^* + \mathrm{RDP} + \mathrm{AIP} + \mathrm{CA} + \mathrm{KFAR}^- \vdash \tau = \tau + \tau^\omega$.

PROOF. As in section 12, 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 10). □

At first sight it seems natural to blame AIP for this violation of liveness, since it identifies the processes $\sum_{n>0} a^n$ and $\sum_{n>0} a^n + a^\omega$, mentioned in section 11, whereas the first one has to terminate eventually while the second one does not have to. However, also for this violation all ingredients of theorem 13 are really needed:

THEOREM 14. *None of the theories of section 13 violates liveness (w.r.t. their own action rules).*

PROOF. In fact all models of section 13 respect liveness (w.r.t. their own action rules). For the models $\mathcal{P}/\leftrightarrows$, $\mathcal{R}/\leftrightarrows$ and $\mathcal{P}/\leftrightarrows_{cl}$ this is trivial, since they are *ft*-consistent (w.r.t. their own action rules). Thus consider the model $\mathbb{G}_{\aleph_0}/\equiv_\Delta$. From section 10 it follows that $\mathbb{G}_{\aleph_0}/\equiv_\Delta$ respects liveness if $\mathbb{G}_{\aleph_0}/\equiv_\Delta \models p = q$ implies that p and q have the same complete traces with δ. Moreover the complete traces of a process, which have another complete trace of that process as initial part, may be skipped for this purpose. So suppose $\mathbb{G}_{\aleph_0}/\equiv_\Delta \models p = q$. Call a complete trace *minimal* if it has no other complete trace as initial part. It has to be proved that p and q have the same minimal complete traces with δ. Let g,h be the process graphs of $p,q \in \mathcal{P}$. Then $\mathcal{F}_\Delta(g) = \mathcal{F}_\Delta(h)$ (see [6]). Thus

- $u\text{-}tr(p) = u\text{-}tr(q)$ (from F1, F2 and F4 in [6]) and hence, using König's lemma, since $\mathbb{G}_{\aleph_0}$ is finitely branching, any infinite trace of p has an initial part in $\omega\text{-}tr(q) \cup \uparrow\text{-}tr(q)$ and vice versa.
- $\surd\text{-}tr(p) = \surd\text{-}tr(q)$ (from F3).
- $\delta\text{-}tr(p) = \delta\text{-}tr(q)$ (from F1, since $\delta\text{-}tr(p) = \{\sigma\delta \mid (\sigma, A \cup \{\surd\}) \in \mathcal{F}_\Delta(g)\}$
- p and q have the same minimal divergence traces (from F4, since the minimal divergence traces

of a process are the same as its minimal subdivergence traces).
Hence p and q have the same minimal complete traces with δ, and $\mathbb{G}_{\aleph_0}/_{\equiv^\cdot}$ respects liveness. □

The problem around the identification of $\sum_{n>0} a^n$ and $\sum_{n>0} a^n + a^\omega$ and the resulting violation of liveness is resolved differently in the mentioned models:
- in $\mathcal{P}/\leftrightarrows$ the processes are not identified, since AIP is not valid.
- in $\mathcal{R}/\leftrightarrows$ the processes do not exist.
- in $\mathcal{P}/\leftrightarrows_d$ only the (closed) process $\sum_{n>0} a^n + a^\omega$ exists and $\sum_{n>0} a^n$ is just an unusual name for $\sum_{n>0} a^n + a^\omega$: also $\sum_{n>0} a^n$ has an a^ω-trace and termination does not have to happen eventually.
- in $\mathbb{G}_{\aleph_0}/_{\equiv_\Delta}$ they are identified, but since both are divergent (see the proof of theorem 10) no good has to happen to any of them.
- in $\mathbb{G}_{\aleph_0}/_{\equiv^\cdot}$ liveness is violated anyway.

15. The validity of AIP$^-$

In this paper a model $\mathcal{P}/\leftrightarrows$ of ACP_τ has been constructed, satisfying RDP, CA and KFAR, but not satisfying AIP. In the sections 12 and 14 it is 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 will be 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 and Klop [2]. In table 3 of the present paper, a simpler and less restrictive version of AIP, called AIP$^-$, is proposed. For this reason the predicates B_n were introduced. Now it remains to be proven that $\mathcal{P}/\leftrightarrows \models \mathrm{AIP}^-$.

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 $\mathbb{G}_{\aleph_1}/_{\leftrightarrows_{rtb}}$, which is isomorphic to $\mathcal{P}/\leftrightarrows$. It makes use of the lemmas of section 7. As a corollary it follows that all rules of table 3 are satisfied by $\mathcal{P}/\leftrightarrows$, and that $\mathrm{ACP}_\tau + \mathrm{PR} + \mathrm{B} + \mathrm{RDP} + \mathrm{AIP}^- + \mathrm{CA} + \mathrm{KFAR}$ is consistent and respects liveness.

Theorem 15. $\mathcal{P}/\leftrightarrows \models \mathrm{AIP}^-$.

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. $S_n \neq \emptyset$, 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 $\bigcap_{n=0}^{\infty} S_n \neq \emptyset$. 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. □

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] 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.

[9] R. MILNER, *A calculus for communicating systems,* Springer LNCS 92, 1980.

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

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

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

[13] 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.