Formal Description of Programming Concepts - III
M. Wirsing (Editor)
Elsevier Science Publishers B.V. (North-Holland)
IFIP, 1987

Conditional axioms and α/β-calculus in process algebra

J.C.M. Baeten,
University of Amsterdam

J.A. Bergstra,
University of Amsterdam,
State University of Utrecht

J.W. Klop,
Centre for Mathematics and Computer Science, Amsterdam

We define the alphabet of finite and infinite terms in ACP_τ, the algebra of communicating processes with silent steps, and also give approximations of it. Using the alphabet, we formulate some conditional axioms. The usefulness of the axioms is demonstrated in examples.

1985 Mathematics Subject Classification: 68Q55, 68Q45, 68Q10, 68N15.
1982 CR Categories: F.1.2, F.3.2, F.3.1, D.3.1.
Key words & Phrases: concurrency, process algebra, alphabet, conditional axiom.
Note: This work was sponsored in part by ESPRIT contract nr. 432, Meteor. This paper is a revised version of reference [1].

1. Introduction.

During the last decade, various process models, or models for concurrency, have been proposed; we mention Milner's synchronisation trees in CCS (see [13]), the metrical process spaces of De Bakker and Zucker [4], the models based on preorders between processes of Hennessy and Plotkin [11] and the failure semantics of Brookes, Hoare and Roscoe [9] and Hoare [12]. (For more complete references we must refer to Bergstra and Klop [6].) Starting with Milner, there has been a growing interest in an *algebraic* treatment of concurrency. Here is our point of departure: rather than fixing a particular process model, we start with axioms describing a class of models. There are

two reasons for this axiomatic methodology. One is that when so many different (but often related) process models arise as the last years have witnessed, it is becoming profitable to attempt an organisation into an axiomatic framework. The second reason is that, being interested in actual specification of processes and verification of process behaviour, we adhere to the principle that one must be able to perform such specifications and verifications exclusively in algebraical terms - rather than in one particular process model. Thus we hope to get, eventually, at a greater manageability of specifications and verifications: ideally, we want pure algebraic formula manipulations, rather than having to resort to particular process representations, such as transition diagrams, Petri nets, or failure sets. This, for the obvious reason that such particular representations are much harder to represent mechanically than equations in an algebraic format, where we can hope to profit from the experience obtained in the culture of abstract data type specifications.

A consequence is that we wish to adhere as much as possible to *equations.* However, as this purely equational medium would not be sufficiently expressive (viz. to prove equations between expressions denoting infinite processes), we will also admit, as is usual in an algebraic/axiomatic style, *proof rules,* or as we will also call them, **conditional axioms**. A typical example of such a proof rule is the Recursive Specification Principle below, stating that if processes p,q satisfy the same (guarded) equation, one may infer the equation p = q.

Not surprisingly, it turns out that in computations with infinite processes one often needs information about the **alphabet** $\alpha(p)$ of a process p. E.g. if p is the process uniquely defined (specified) by the recursion equation $x = a \cdot x$ (where 'a' is an atomic action), we have $\alpha(p) = \{a\}$. An example of the use of alphabet information is given by:

$$\alpha(x) \cap I = \emptyset \Rightarrow \tau_I(x) = x \qquad \text{(CA4)}$$

in words: if no action from I occurs in process x, then hiding (abstracting from) actions of I in x has no effect. A more interesting axiom is conditional axiom CA2, which allows one to commute abstraction and parallel composition (in appropriate circumstances); it is vital for the verification of systems with three or more components put in parallel.

In order to attach a precise meaning to conditional axioms of the above form we need some insight in the notion of the alphabet $\alpha(p)$ of process p. We assume that p has been specified by means of a recursive specification with guarded recursion. Then one can effectively find sets $\alpha_1(p), \alpha_2(p), \alpha_3(p), \ldots$, and $\beta_1(p), \beta_2(p), \beta_3(p), \ldots$, such that

$$\alpha_1(p) \subseteq \alpha_2(p) \subseteq \alpha_3(p) \subseteq \ldots\ldots \subseteq \alpha(p)$$

$$\beta_1(p) \supseteq \beta_2(p) \supseteq \beta_3(p) \supseteq \ldots\ldots \supseteq \alpha(p)$$

(see 3.2, 4.5). In general $\bigcup_{n\geq 1} \alpha_n(p) = \alpha(p)$ but $\bigcap_{n\geq 1} \beta_n(p) = \alpha(p)$ need not hold

(this is connected with the fact that on the basis of a given recursive specification of p the alphabet $\alpha(p)$ cannot in general be effectively computed, see 3.5). In practical cases, either one finds n,m such that $\alpha_m(p) = \beta_n(p)$ $(=\alpha(p))$, or $\beta_n(p)$ is sufficiently small to verify the condition of a conditional axiom. We call this small theory about alphabets

the α/β-calculus. Though very simple, this α/β-calculus seems to be an indispensable tool in system verification based on process algebra (as we will call an algebraic framework such as the one presented below).
In the last sections we describe three examples of simple system verifications which extensively demonstrate the use of the conditional axioms CA1-7.
System verifications of a related nature, but performed in a model, can be found in Sifakis [15] and Olderog [14].
The consistency of CA1-7 on top of ACP_τ + KFAR + RSP is a nontrivial issue. We refer to [3] for such a consistency proof. In this paper, we just check each of the laws in detail for all finite processes. Here RSP is the Recursive Specification Principle, already mentioned above (see 2.9), and KFAR is Koomen's Fair Abstraction Rule, explained in 3.4. There, we find a rather unexpected connection between KFAR and determination of alphabets.

2. Algebra of communicating processes with silent steps.

2.1 The axiomatic framework in which we present this document is ACP_τ, the algebra of communicating processes with silent steps, as described in Bergstra & Klop [7]. In this section, we give a brief review of ACP_τ.
Process algebra starts from a finite collection A of given objects, called atomic actions, atoms or steps. These actions are taken to be indivisible, usually have no duration and form the basic building blocks of our systems. The first two compositional operators we consider are $\cdot$, denoting sequential composition, and + for alternative composition. If x and y are two processes, then $x \cdot y$ is the process that starts the execution of y after the completion of x, and x+y is the process that chooses either x or y and executes the chosen process. Each time a choice is made, we choose from a set of alternatives. We do not specify whether the choice is made by the process itself, or by the environment. Axioms A1-5 in table 1 below give the laws that + and $\cdot$ obey. We leave out $\cdot$ and brackets as in regular algebra, so $xy + z$ means $(x \cdot y) + z$.
On intuitive grounds $x(y + z)$ and $xy + xz$ present different mechanisms (the moment of choice is different), and therefore, an axiom $x(y + z) = xy + xz$ is not included.
We have a special constant δ denoting deadlock, the acknowledgement of a process that it cannot do anything anymore, the absence of an alternative. Axioms A6,7 give the laws for δ.
Next, we have the parallel composition operator $\|$, called merge. The merge of processes x and y will interleave the actions of x and y, except for the communication actions. In $x\|y$, we can either do a step from x, or a step from y, or x and y both synchronously perform an action, which together make up a new action, the communication action. This trichotomy is expressed in axiom CM1. Here, we use two auxiliary operators $\lfloor\!\lfloor$ (left-merge) and $|$ (communication merge). Thus, $x\lfloor\!\lfloor y$ is $x\|y$, but

with the restriction that the first step comes from x, and $x|y$ is $x\|y$ with a communication step as the first step. Axioms CM2-9 give the laws for $\llcorner$ and $|$. On atomic actions, we assume the communication function given, obeying laws C1-3. Finally, we have on the left-hand side of table 1 the laws for the encapsulation operator ∂_H. Here H is a set of atoms, and ∂_H blocks actions from H, renames them into δ. The operator ∂_H can be used to encapsulate a process, i.e. to block communications with the environment.
The right-hand side of table 1 is devoted to laws for Milner's silent step τ (see [13]). Laws T1-3 are Milner's τ-laws, and TM1,2 and TC1-4 describe the interaction of τ and merge. Finally, τ_I is the abstraction operator, that renames atoms from I into τ.
In table 1 we have $a,b,c \in A_\delta$ (i.e. $A \cup \{\delta\}$), x,y,z are arbitrary processes, and $H,I \subseteq A$.

$x + y = y + x$	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		
$\delta\|a = \delta$	C3		
$x\|y = x\llcorner y + y\llcorner x + x\|y$	CM1		
$a\llcorner x = ax$	CM2	$\tau\llcorner x = \tau x$	TM1
$ax\llcorner y = a(x\|y)$	CM3	$\tau x\llcorner y = \tau(x\|y)$	TM2
$(x + y)\llcorner z = x\llcorner z + y\llcorner z$	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

Table 1. ACP_τ.

2.2 Definition: The set of **basic terms**, BT, is inductively defined as follows:
i. $\tau, \delta \in BT$ ii. if $t \in BT$, then $\tau t \in BT$
iii. if $t \in BT$ and $a \in A$, then $at \in BT$ iv. if $t, s \in BT$, then $t+s \in BT$.

2.3 Elimination theorem: (Bergstra & Klop [7]) Let t be a closed term over ACP_τ. Then there is a basic term s such that $ACP_\tau \vdash t=s$.

2.4 Theorem 2.3 allows us to use induction in proofs. The set of closed terms modulo derivability (the initial algebra) forms a model for ACP_τ. However, most processes encountered in practice cannot be represented by a closed term, but will be specified recursively. Therefore, most models of process algebra also contain infinite processes, that can be recursively specified. First, we develop some terminology.

2.5 Definitions: i) Let t be a term over ACP_τ, and x a variable in t. Suppose that the abstraction operator τ_I does not occur in t. Then we say that an occurrence of x in t is **guarded** if t has a subterm of the form $a \cdot s$, with $a \in A_\delta$ (so $a \neq \tau$!) and this x occurs in s. (I.e. each variable is *preceded* by an atom.)
ii) A **recursive specification** over ACP_τ is a set of equations $\{x = t_x : x \in X\}$, with X a set of variables, and t_x a term over ACP_τ and variables X (for each $x \in X$). No other variables may occur in t_x.
iii) A recursive specification $\{x = t_x : x \in X\}$ is **guarded** if no t_x contains an abstraction operator τ_I, and each occurrence of a variable in each t_x is guarded.

2.6 Notes: i) The constant τ cannot be a guard, since the presence of a τ does not lead to unique solutions: to give an example, the equation $x = \tau x$ has each process starting with a τ as a solution.
ii) A definition of guardedness involving τ_I is very complicated, and therefore, we do not give such a definition here. The definition above suffices for our purposes.

2.7 Definition: On ACP_τ, we can define a **projection operator** π_n, that cuts off a process after n atomic steps are executed, by the axioms in table 2 ($n \geq 1$, $a \in A_\delta$, x,y are arbitrary processes).

$\pi_n(a) = a$	$\pi_n(\tau) = \tau$
$\pi_1(ax) = a$	$\pi_n(\tau x) = \tau \cdot \pi_n(x)$
$\pi_{n+1}(ax) = a \cdot \pi_n(x)$	
$\pi_n(x + y) = \pi_n(x) + \pi_n(y)$	

Table 2. Projection.

Remarks: Because of the τ-laws, we must have that executing a τ does not increase depth. A process p is **finite** if it is equal to a closed term; otherwise p is **infinite**. Note that if p is finite, there is an n such that $\pi_n(p) = p$.

2.8 Theorem: If the set of processes P forms a solution for a guarded recursive specification E, then $\pi_n(p)$ is equal to some closed ACP_τ-term for each $p \in P$ and $n \geq 1$, and this term does not depend on the particular solution P.
Proof: Let E^n be the **n-th expansion** of E, i.e. the recursive specification obtained by substituting terms t_x for variables x occurring in the right-hand sides of its equations, repeating this procedure n times. Since E is guarded, we see that in the n-th expansion of E, each variable is n times guarded. Since the set P is a solution of E, and E^n is obtained by substitution, P is also a solution of E^n. Now we calculate $\pi_n(p)$ using the equation for p in E^n, with the axioms in table 3 above. We see that $\pi_n(p)$ does not depend on which processes we substituted in the right-hand side of this equation: since each variable was n times guarded, the calculation stops before the variable is reached. It is easy to finish the proof.

2.9 Theorem 2.8 leads us to formulate the following two principles, which together imply that each guarded recursive specification has a unique solution (determined by its finite projections).
The **Recursive Definition Principle (RDP)** is the assumption that each guarded recursive specification has at least one solution, and the **Recursive Specification Principle (RSP)** is the assumption that each guarded recursive specification has at most one solution. In this paper, we assume RDP and RSP (for more about these principles, see [3]).
To give an example, if p is a solution of the guarded recursive specification $\{x = a \cdot x\}$, we find $\pi_n(p) = a^n$ for all $n \geq 1$, so we can put $p = a^\omega$. For more information, see [3].
Abusing language, we also use the variables in a guarded recursive specification for the process that is its unique solution.

2.10 In Baeten, Bergstra & Klop [3], a model is presented for ACP_τ, consisting of rooted, directed multigraphs, with edges labeled by elements of $A \cup \{\delta,\tau\}$, modulo a congruence relation called rooted $\tau\delta$-bisimulation (comparable to Milner's observational congruence, see [13]). In this model all axioms presented in this paper hold, and also principles RDP and RSP hold.
Moreover, each element in this model can either be specified by a guarded recursive specification, or can be found from such a process by abstraction (by an application of the operator τ_I).

2.11 The axioms of **Standard Concurrency** (displayed in table 3, on the following page) will also be used in the sequel. A proof that they hold for all closed terms can be

found in Bergstra & Klop [7].

$(x \mathbin{\lfloor\!\lfloor} y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} (y \| z)$
$(x \mid ay) \mathbin{\lfloor\!\lfloor} z = x \mid (ay \mathbin{\lfloor\!\lfloor} z)$
$x \mid y = y \mid x$
$x \| y = y \| x$
$x \mid (y \mid z) = (x \mid y) \mid z$
$x \| (y \| z) = (x \| y) \| z$

Table 3. Standard concurrency.

3. Alphabets.

3.1 Definition: The **alphabet** of a process is the set of atomic actions that it can perform, so is a subset of A. In order to define the alphabet function α on closed terms, we have the axioms in table 4 ($a \in A$, x,y are arbitrary processes).

$\alpha(\delta) = \emptyset$	AB1
$\alpha(\tau) = \emptyset$	AB2
$\alpha(ax) = \{a\} \cup \alpha(x)$	AB3
$\alpha(\tau x) = \alpha(x)$	AB4
$\alpha(x + y) = \alpha(x) \cup \alpha(y)$	AB5

Table 4. Alphabet.

Note that $\alpha(\delta) = \alpha(\tau) = \emptyset$ is necessary by axioms A6 and T1.

3.2 Now we want to define α on infinite processes.
We define the alphabet for solutions of guarded recursive specifications (which are in general infinite processes) by adding the following axiom to table 4:

$$\alpha(x) = \bigcup_{n \geq 1} \alpha(\pi_n(x)) \qquad \text{AB6.}$$

By theorem 2.8, each $\pi_n(x)$ is a closed term, if x is the solution of a guarded recursive specification, so $a(\pi_n(x))$ can be determined with the axioms in table 4. It is not hard to see that equation AB6 holds for all closed terms (using structural induction), so this axiom does not contradict axioms AB1-5. Further note, that since the partial unions $\alpha(\pi_1(x)) \cup \ldots \cup \alpha(\pi_n(x))$ form an increasing sequence (as $n \to \infty$), and the set of alphabets is finite (since A is finite), the sequence will be eventually constant, and the limit will always exist.

3.3 We still have not defined the alphabet for all processes that we want to consider. To give an example, if x is given by the guarded recursive specification $\{x = ax\}$ (we say $x = a^\omega$), put $y = \tau_{\{a\}}(x)$, and then the process y cannot be given by a guarded recursive specification (y is the process τ^ω). To define the alphabet of such processes, found by abstraction from certain actions in a process given by a guarded recursive specification, we add one more axiom to table 4:

$$\alpha(\tau_I(x)) = \alpha(x) - I \qquad \text{AB7.}$$

Again, it is clear that this axiom holds for all closed terms.

3.4 Example: Let x be given by $\{x = ax\}$, and define $y = \tau_{\{a\}}(x)$, $z = y \cdot b$ (with $b \neq a$). What is the alphabet of z? Well, we know $\alpha(x) = \{a\}$ (for $\pi_n(x) = a^n$ for each $n \geq 1$), so $\alpha(y) = \alpha(x) - \{a\} = \emptyset$. Then, $\alpha(z) = \alpha(y \cdot b) = \alpha(\tau_{\{a\}}(x) \cdot b) = \alpha(\tau_{\{a\}}(x) \cdot \tau_{\{a\}}(b)) = \alpha(\tau_{\{a\}}(x \cdot b)) = \alpha(x \cdot b) - \{a\} = \emptyset$, for $\pi_n(xb) = a^n$ for each $n \geq 1$.

We can motivate this result in a different way, if we use Koomen's Fair Abstraction Rule (KFAR, see [3], and Vaandrager [16]):

$$x = i \cdot x + y,\ i \in I \Rightarrow \tau_I(x) = \tau \cdot \tau_I(y) \qquad \text{KFAR.}$$

KFAR expresses the fact that, due to some fairness mechanism, i (usually some *internal* action) resists being performed infinitely many times consecutively. Here, we have $x = a \cdot x = a \cdot x + \delta$, so by KFAR $y = \tau_{\{a\}}(x) = \tau\delta$. Then $z = yb = \tau\delta b = \tau\delta$, and we see again that $\alpha(y) = \alpha(z) = \emptyset$.

3.5 Theorem: it is in general *undecidable,* to which set $\alpha(x)$ is equal.

Proof: Let K be a recursively enumerable, but not recursive subset of $\mathbb{N}$ (the set of natural numbers). In Bergstra & Klop [5] a recursive specification, parametrised by $n \in \mathbb{N}$, over finitely many variables $x_1,\ldots,x_k$ is given (k depends on K), such that we have the following: $x_1(n) = b^\omega$ if $n \notin K$

$x_1(n) = b^m \cdot \text{stop}$ if $n \in K$ (for some $m \in \mathbb{N}$)

(here b, stop are atomic actions). Thus we have $\alpha(x_1(n)) = \{b\}$ if $n \notin K$ and $\alpha(x_1(n)) = \{b,\text{stop}\}$ if $n \in K$. Since K is not recursive, determining whether $n \in K$, for a given n, is undecidable, so determining $\alpha(x_1)$ is undecidable.

4. α/β-calculus and conditional axioms.

4.1 Axiom AB6 in 3.2 gives a sequence of subsets of $\alpha(x)$, which will converge to $\alpha(x)$. However, as we remarked, finding $\alpha(x)$ itself can sometimes be very difficult. Luckily, in applications it is often sufficient to have a superset of $\alpha(x)$, which is not too big. With such a superset, we can even determine $\alpha(x)$ in many cases. For this reason, we define $\beta(x)$ in 4.4. First we need theorem 4.2. A piece of notation: if $B,C \subseteq A \cup \{\delta,\tau\}$, we define $B \mid C = \{b \mid c : b \in B, c \in C\} - \{\delta\}$ (we leave out δ, so that $B \mid C$ is an alphabet).

4.2 Theorem: The following hold for all closed ACP_τ-terms t,s:

i. $\alpha(t \cdot s) \subseteq \alpha(t) \cup \alpha(s)$

ii. $\alpha(t \| s) = \alpha(t) \cup \alpha(s) \cup \alpha(t) \mid \alpha(s)$

iii. $\alpha(t \lfloor\!\lfloor s) \subseteq \alpha(t \| s)$

iv. $\alpha(t \mid s) \subseteq \alpha(t \| s)$

v. $\alpha(\partial_H(t)) \subseteq \alpha(t) - H$

Proof: By theorem 2.3, we only have to prove these statements for all basic terms.

i. We use induction on the structure of t, as defined in 2.2.

Case 1: if $t = \tau$, $\alpha(ts) = \alpha(\tau s) = \alpha(s) = \alpha(\tau) \cup \alpha(s)$; if $t = \delta$, $\alpha(ts) = \alpha(\delta s) = \alpha(\delta) = \emptyset \subseteq \alpha(\delta) \cup \alpha(s)$.

Case 2: if $t = \tau t'$, then $\alpha(ts) = \alpha(\tau t's) = \alpha(t's) \subseteq \alpha(t') \cup \alpha(s)$ (by induction hypothesis) $= \alpha(\tau t') \cup \alpha(s)$.

Case 3: if $t = at'$ ($a \in A$), then $\alpha(ts) = \alpha(at's) = \{a\} \cup \alpha(t's) \subseteq \{a\} \cup \alpha(t') \cup \alpha(s)$ (by induction hypothesis) $= \alpha(at') \cup \alpha(s)$.

Case 4: if $t = t' + t''$, then $\alpha(ts) = \alpha((t' + t'')s) = \alpha(t's + t''s) = \alpha(t's) \cup \alpha(t''s) \subseteq \alpha(t') \cup \alpha(s) \cup \alpha(t'') \cup \alpha(s)$ (by induction hypothesis) $= \alpha(t' + t'') \cup \alpha(s)$.

ii. This is more complicated. We do simultaneous induction on t and s, and write

$$t = \sum_{1 \le i \le I} a_i t_i + \sum_{1 \le j \le J} \tau t'_j + (\tau) + \delta,$$

$$s = \sum_{1 \le k \le K} b_k s_k + \sum_{1 \le n \le N} \tau s'_n + (\tau) + \delta$$

with $I,J,K,N \ge 0$, $a_i, b_k \in A$, and the single τ may or may not occur. By induction hypothesis we can assume that ii holds for all terms

$$t \| s_k,\ t \| s'_n,\ t_i \| s,\ t_i \| s_k,\ t_i \| s'_n,\ t'_j \| s,\ t'_j \| s_k,\ t'_j \| s'_n.$$

To expand $t \| s$, we use the rules of table 1.

$$t \| s = \delta + \Sigma\, a_i(t_i \| s) + \Sigma\, \tau(t'_j \| s) + (\tau s) +$$
$$+ \Sigma\, b_k(t \| s_k) + \Sigma\, \tau(t \| s'_n) + (\tau t) + \Sigma\, (a_i \mid b_k)(t_i \| s_k)$$
$$+ \Sigma\, a_i t_i \mid \tau s'_n + \Sigma\, \tau t'_j \mid b_k s_k + \Sigma\, \tau t'_j \mid \tau s'_n.$$

Now the three summands on the last line can be skipped, since they are summands of other terms (for instance, each $a_i t_i \mid \tau s'_n = a_i t_i \mid s'_n$ is a summand of $\tau(t \| s'_n)$, see Bergstra & Klop [7], 3.6). Next we use definition 3.1, and obtain:

$$\alpha(t \| s) = \bigcup_{i \in I} \{a_i\} \cup \alpha(t_i \| s) \cup \bigcup_{j \in J} \alpha(t'_j \| s) \cup (\alpha(s)) \cup \bigcup_{k \in K} \{b_k\} \cup \alpha(t \| s_k) \cup$$
$$\bigcup_{n \in N} \alpha(t \| s'_n) \cup (\alpha(t)) \cup \bigcup_{i,k \text{ with } a_i \mid b_k \neq \delta} \{a_i \mid b_k\} \cup \alpha(t_i \| s_k).$$

Now we apply the induction hypothesis, and obtain

$$\alpha(t \| s) = \bigcup_{i \in I} \{a_i\} \cup \alpha(t_i) \cup \alpha(s) \cup \alpha(t_i) \mid \alpha(s) \cup \bigcup_{j \in J} \alpha(t'_j) \cup \alpha(s) \cup \alpha(t'_j) \mid \alpha(s) \cup$$
$$\bigcup_{k \in K} \{b_k\} \cup \alpha(t) \cup \alpha(s_k) \cup \alpha(t) \mid \alpha(s_k) \cup \bigcup_{n \in N} \alpha(t) \cup \alpha(s'_n) \cup \alpha(t) \mid \alpha(s'_n)$$

$$\bigcup_{i,k \text{ with } a_i \mid b_k \neq \delta} \{a_i \mid b_k\} \cup \alpha(t_i) \cup \alpha(s_k) \cup \alpha(t_i) \mid \alpha(s_k) =$$

$$= \bigcup_{i \in I} \{a_i\} \cup \alpha(t_i) \cup \bigcup_{j \in J} \alpha(t'_j) \cup \alpha(t) \cup \bigcup_{k \in K} \{b_k\} \cup \alpha(s_k) \cup \bigcup_{n \in N} \alpha(s'_n) \cup \alpha(s) \cup$$

$$\bigcup_{i \in I} \alpha(t_i) \mid \alpha(s) \cup \bigcup_{j \in J} \alpha(t'_j) \mid \alpha(s) \cup \bigcup_{k \in K} \alpha(t) \mid \alpha(s_k) \cup \bigcup_{n \in N} \alpha(t) \mid \alpha(s'_n) \cup$$

$$\bigcup_{i,k \text{ with } a_i \mid b_k \neq \delta} \{a_i \mid b_k\} \cup \alpha(t_i) \mid \alpha(s_k).$$

Now it is not hard to see, that the union of the first part of the first line is $\alpha(t)$, of the second part $\alpha(s)$, and the union of the last two lines is $\alpha(t) \mid \alpha(s)$. This finishes the proof of statement ii.

iii, iv: these follow immediately from axioms CM1 and AB5.

v: We use induction on the structure of t, as given in 2.2.

Case 1: if $t=\delta$, $\alpha(\partial_H(t)) = \alpha(\partial_H(\delta)) = \alpha(\delta) = \alpha(\delta) - H$, if $t=\tau$, $\alpha(\partial_H(t)) = \alpha(\partial_H(\tau)) = \alpha(\tau) = \alpha(\tau) - H$.

Case 2: if $t=\tau t'$, $\alpha(\partial_H(t)) = \alpha(\partial_H(\tau t')) = \alpha(\partial_H(t')) \subseteq \alpha(t') - H$ (by induction hypothesis) $= \alpha(\tau t') - H$.

Case 3: if $t=at'$, and $a \in H$, we have $\alpha(\partial_H(t)) = \alpha(\partial_H(at')) = \alpha(\delta \cdot \partial_H(t')) = \alpha(\delta) = \emptyset \subseteq \alpha(at') - H$; if $a \notin H$, $\alpha(\partial_H(t)) = \alpha(\partial_H(at')) = \alpha(a \cdot \partial_H(t')) = \{a\} \cup \alpha(\partial_H(t')) \subseteq \{a\} \cup (\alpha(t') - H)$ (by induction hypothesis) $= (\{a\} \cup \alpha(t')) - H = \alpha(at') - H$.

This finishes the proof of theorem 4.2.

Remark: In the sequel we will assume that theorem 4.2 holds for *all* processes x,y. We refer to [3] for a proof that it is consistent to do so.

4.3 Definition: suppose $t(x_1,....,x_n)$ is an ACP_τ-term with variables $x_1,....,x_n$. We define a set-term corresponding to t, involving the alphabets of these variables. We do that by applying the rules in 3.1, 3.3 and 4.2 to $\alpha(t)$, working from the outside in. We go on until we only have unknowns $\alpha(x_j)$ left, so α is not applied to any composite term. We obtain $\alpha(t) \subseteq t^*(\alpha(x_1),.....,\alpha(x_n))$, where t^* is a term over the signature with as sort the powerset of A, as functions set union $\cup$, set difference -H (a unary operator for each H appearing as a subscript in a ∂_H or τ_H), and communication $\mid$ (as defined in 4.1), and as constants {a} for each $a \in A$, and $\emptyset$.

Example: if $t \equiv a \cdot x_4(x_1 \| x_2 + \partial_H(x_3))$, then $t^* \equiv \{a\} \cup \alpha(x_4) \cup \alpha(x_1) \cup \alpha(x_2) \cup \alpha(x_1) \mid \alpha(x_2) \cup (\alpha(x_3) - H)$.

4.4 Definition: Let $\{x = t_x : x \in X\}$ be a guarded recursive specification, and let $x \in X$. Suppose t_x contains variables $x_1,....,x_n$. Then define $\beta(x)$ to be least fixed point of the equation

$$\beta(x) = t_x^*(\beta(x_1),.....,\beta(x_n)).$$

Note that this least fixed point will always exist, since terms t^* over (Pow(A), $\cup$, -H, $\mid$, {a}) are *monotonic* (i.e. a relation $X \subseteq Y$ is preserved under the operations). Thus, $\beta(x)$

is the limit of successive approximations $t_x^*(\emptyset,....,\emptyset)$, $t_x^*(t_{x1}^*(\emptyset,...,\emptyset),.....,t_{xn}^*(\emptyset,...,\emptyset))$, etc..

4.5 Theorem: Let $E = \{x = t_x : x \in X\}$ be a guarded recursive specification and let $x \in X$. Then $\alpha(x) \subseteq \beta(x)$.

Proof: Let E^n be the n-th expansion of E, as defined in 2.8. Let $\beta_n(x)$ be the $\beta(x)$ belonging to the equation for x in E^n.

We claim that then $\beta(x) \supseteq \beta_n(x)$, for each $x \in X$. Let $t^n{}_x$ be the right-hand side of the equation for x in E^n. To prove the claim, first take n=2. Suppose $t^2{}_x$ has variables $x_1,...,x_k$. Then $t^2{}_x^*(\beta(x_1), ..., \beta(x_k)) = t_x^*(t_{x1}^*(\beta(x_{1,1}), ...), ..., t_{xk}^*(\beta(x_{k,1}), ...)) =$
$= t_x^*(\beta(x_1), ..., \beta(x_k)) = \beta(x)$ (for certain variables $x_{i,j}$).

Therefore, $\beta(x)$ is a fixed point of equation $x = t^2{}_x^*(x_1, ..., x_k)$. Since $\beta_2(x)$ is the *least* fixed point of this equation, we must have $\beta(x) \supseteq \beta_2(x)$. The general case follows by iteration. This proves the claim.

By theorem 2.8, $\pi_n(x)$ is equal to some closed term, which is independent of the processes substituted for the variables in t_x. Therefore

$$\alpha(\pi_n(x)) = \alpha(\pi_n(t^n{}_x)) = \alpha(\pi_n(t^n{}_x(x_1,...,x_k))) = \alpha(\pi_n(t^n{}_x(\delta,...,\delta))) \subseteq$$
$$\subseteq \alpha(t^n{}_x(\delta,...,\delta)) \subseteq t^n{}_x^*(\emptyset,...,\emptyset) \subseteq \beta_n(x) \subseteq \beta(x),$$

and with axiom AB6 it follows that $\alpha(x) \subseteq \beta(x)$.

4.6 Notes: i) It can be shown that the fixed point $\beta(x)$ can be reached from $\emptyset$ in finitely many iterations, if we assume that the specification is finite and assume the Handshaking Axiom $a|b|c = \delta$ (the Handshaking Axiom says that only two-way communications can occur; assuming it holds ensures that all possible communication actions are generated the first time).

ii) We cannot have in general that $\bigcap_{n\geq 1} \beta_n(x) = \alpha(x)$, because that would make the determination of $\alpha(x)$ decidable, contradicting 3.5.

4.7 Example: A bag B^{ij} with input port i and output port j ($i \neq j$) is given by the guarded recursive specification

$$B^{ij} = \sum_{d\in D} ri(d)\cdot(sj(d)\|B^{ij}) \qquad \text{(see Bergstra \& Klop [5]).}$$

Here D is a finite set of data, ri(d) means **receive** d along i, and sj(d) means **send** d along j. We find $\alpha(\pi_2(B^{ij})) = \alpha(\sum_{d\in D} ri(d)[sj(d) + \sum_{e\in D} ri(e)]) = \{ri(d),sj(d) : d\in D\}$, and on the other hand $\alpha(B) = \{ri(d),sj(d) : d\in D\} \cup \alpha(B) \cup \{sj(d) : d\in D\} | \alpha(B)$ using 4.2, so $\beta(B) = \{ri(d),sj(d) : d\in D\}$. Since $\alpha(\pi_2(B)) \subseteq \alpha(B) \subseteq \beta(B)$, we must have $\alpha(B) = \{ri(d),sj(d) : d\in D\}$. In this way we can calculate the alphabets of many interesting specifications. In the proofs of the following theorems, extensive use is made of this so-called α/β-calculus.

4.8 In the following table 4, we present 7 conditional axioms:

$\alpha(x) \mid (\alpha(y) \cap H) \subseteq H$	$\Rightarrow$	$\partial_H(x \| y) = \partial_H(x \| \partial_H(y))$	CA1
$\alpha(x) \mid (\alpha(y) \cap I) = \emptyset$	$\Rightarrow$	$\tau_I(x \| y) = \tau_I(x \| \tau_I(y))$	CA2
$\alpha(x) \cap H = \emptyset$	$\Rightarrow$	$\partial_H(x) = x$	CA3
$\alpha(x) \cap I = \emptyset$	$\Rightarrow$	$\tau_I(x) = x$	CA4
$H = J \cup K$	$\Rightarrow$	$\partial_H(x) = \partial_J \circ \partial_K(x)$	CA5
$I = J \cup K$	$\Rightarrow$	$\tau_I(x) = \tau_J \circ \tau_K(x)$	CA6
$H \cap I = \emptyset$	$\Rightarrow$	$\tau_I \circ \partial_H(x) = \partial_H \circ \tau_I(x)$	CA7

Table 4. Conditional axioms.

4.9 Theorem: Axioms CA1-7 hold for all closed ACP_τ-terms.

Proof: By theorem 2.3, we only have to prove these statements for all basic terms t,s.

CA1: We do simultaneous induction on t and s, as in the proof of 4.2, and write

$$t = \sum_{1 \le i \le I} a_i t_i + \sum_{1 \le j \le J} \tau t'_j + (\tau) + \delta,$$

$$s = \sum_{1 \le k \le K} b_k s_k + \sum_{1 \le m \le M} h_m s''_m + \sum_{1 \le n \le N} \tau s'_n + (\tau) + \delta$$

with $I,J,K,M,N \ge 0$, $a_i \in A$, $b_k \in A\text{-}H$, $h_m \in H$, and the single τ may or may not occur. Now $a_i \in \alpha(t)$ and $h_m \in \alpha(y) \cap H$, so by assumption $a_i \mid h_m \in H$ for each $i \le I$, $m \le M$. Now

$$\partial_H(t \| s) = (\tau) + \delta + \Sigma\, \partial_H(a_i) \cdot \partial_H(t_i \| s) + \Sigma\, \tau \cdot \partial_H(t'_j \| s) +$$
$$+ \Sigma\, b_k \cdot \partial_H(t \| s_k) + \Sigma\, \tau \cdot \partial_H(t \| s'_n) + \Sigma\, \partial_H(a_i \mid b_k) \cdot \partial_H(t_i \| s_k) +$$
$$+ \Sigma\, \partial_H(a_i t_i \mid \tau s'_n) + \Sigma\, \partial_H(\tau t'_j \mid b_k s_k) + \Sigma\, \partial_H(\tau t'_n \mid \tau s'_n).$$

As in the proof of 4.2, we see that we can omit the last three summands. Now we apply the induction hypothesis, which is possible since $\alpha(t_i)$, $\alpha(t'_j) \subseteq \alpha(t)$ and $\alpha(s_k)$, $\alpha(s'_n) \subseteq \alpha(s)$. We obtain

$$\partial_H(t \| s) = (\tau) + \delta + \Sigma\, \partial_H(a_i) \cdot \partial_H(t_i \| \partial_H(s)) + \Sigma\, \tau \cdot \partial_H(t'_j \| \partial_H(s)) +$$
$$+ \Sigma\, b_k \cdot \partial_H(t \| \partial_H(s_k)) + \Sigma\, \tau \cdot \partial_H(t \| \partial_H(s'_n)) + \Sigma\, \partial_H(a_i \mid b_k) \cdot \partial_H(t_i \| \partial_H(s_k)).$$

We use the same argument to add the terms

$$\Sigma\, \partial_H(a_i t_i \mid \tau \cdot \partial_H(s'_n)) + \Sigma\, \partial_H(\tau t'_j \mid b_k \cdot \partial_H(s_k)) + \Sigma\, \partial_H(\tau t'_j \mid \tau \cdot \partial_H(s'_n)).$$

Then we see that the sum of the last two expressions is $\partial_H(t \| \partial_H(s))$, and the proof is finished.

CA2: the proof that CA2 holds for all closed terms is entirely similar to the proof of CA1. Note that now we have to have $a_i \mid h_m = \delta$ (if $a_i \in \alpha(t)$, $h_m \in \alpha(s) \cap I$), so that $\tau_I(a_i t_i \mid h_m s''_m) = \tau_I((a_i \mid h_m)(t_i \| s''_m)) = \delta$, and all these terms drop out.

CA3: this is by induction on the structure of t. We have four cases:

Case 1: $t = \delta$ or τ. Immediate.

Case 2: if $t = \tau t'$, we have by assumption $\emptyset = \alpha(t) \cap H = \alpha(t') \cap H$, so $\partial_H(t) = \partial_H(\tau t') = \tau \cdot \partial_H(t') = \tau t' = t$.

Case 3: if $t = at'$ ($a \in A$), we have by assumption $\emptyset = \alpha(t) \cap H = (\{a\} \cup \alpha(t')) \cap H$, so $a \notin H$

and $\alpha(t') \cap H = \emptyset$, whence $\partial_H(at') = \partial_H(a)\partial_H(t') = at' = t$.
Case 4: if $t = t' + t''$, we have by assumption $\emptyset = \alpha(t) \cap H = (\alpha(t') \cup \alpha(t'')) \cap H$, so $\alpha(t') \cap H = \emptyset$ and $\alpha(t'') \cap H = \emptyset$. Then $\partial_H(t) = \partial_H(t' + t'') = \partial_H(t') + \partial_H(t'') = t' + t'' = t$.
CA4: entirely similar to the proof of CA3.
CA5: by induction on the structure of t. We have four cases:
Case 1: $t = \delta$ or τ. Immediate.
Case 2: if $t = \tau t'$, $\partial_H(t) = \partial_H(\tau t') = \tau \cdot \partial_H(t') = \partial_J \circ \partial_K(\tau) \cdot \partial_J \circ \partial_K(t')$ (induction hypothesis) $= \partial_J \circ \partial_K(\tau t') = \partial_J \circ \partial_K(t)$.
Case 3: if $t = at'$ and $a \notin H$, then also $a \notin J$ and $a \notin K$. Thus $\partial_H(t) = \partial_H(at') = a \cdot \partial_H(t') = \partial_J \circ \partial_K(a) \cdot \partial_J \circ \partial_K(t')$ (induction hypothesis) $= \partial_J \circ \partial_K(at') = \partial_J \circ \partial_K(t)$; if $a \in H$, we have two cases:
Case 3.1: $a \in K$. Then $\partial_H(t) = \partial_H(at') = \delta = \partial_J(\delta) = \partial_J \circ \partial_K(at') = \partial_J \circ \partial_K(t)$.
Case 3.2: Otherwise. Then $a \in J\text{-}K$, so $\partial_H(t) = \partial_H(at') = \delta = \partial_J(a \cdot \partial_K(t')) =$
$= \partial_J(\partial_K(a) \cdot \partial_K(t')) = \partial_J \circ \partial_K(at') = \partial_J \circ \partial_K(t)$.
Case 4: if $t = t' + t''$, $\partial_H(t) = \partial_H(t' + t'') = \partial_H(t') + \partial_H(t'') = \partial_J \circ \partial_K(t') + \partial_J \circ \partial_K(t'')$ (induction hypothesis) $= \partial_J(\partial_K(t') + \partial_K(t'')) = \partial_J \circ \partial_K(t' + t'') = \partial_J \circ \partial_K(t)$.
CA6: similar to CA5.
CA7: by induction on the structure of t. We have four cases:
Case 1: $t = \delta$ or τ. Immediate.
Case 2: if $t = \tau t'$, $\tau_I \circ \partial_H(t) = \tau_I \circ \partial_H(\tau t') = \tau \cdot \tau_I \circ \partial_H(t') = \tau \cdot \partial_H \circ \tau_I(t')$ (induction hypothesis) $= \partial_H \circ \tau_I(\tau t') = \partial_H \circ \tau_I(t)$.
Case 3: if $t = at'$ ($a \in A$), we consider three subcases:
Case 3.1: $a \notin I$, $a \notin H$. Then $\tau_I \circ \partial_H(t) = \tau_I \circ \partial_H(at') = a \cdot \tau_I \circ \partial_H(t') = a \cdot \partial_H \circ \tau_I(t')$ (induction hypothesis) $= \partial_H \circ \tau_I(at') = \partial_H \circ \tau_I(t)$.
Case 3.2: $a \notin I$, $a \in H$. Then $\tau_I \circ \partial_H(t) = \tau_I \circ \partial_H(at') = \tau_I(\delta) = \delta = \partial_H(a \cdot \tau_I(t')) =$
$= \partial_H(\tau_I(a) \cdot \tau_I(t')) = \partial_H \circ \tau_I(at') = \partial_H \circ \tau_I(t)$.
Case 3.3: $a \in I$, $a \notin H$. Then $\tau_I \circ \partial_H(t) = \tau_I \circ \partial_H(at') = \tau_I(a \cdot \partial_H(t')) = \tau \cdot \tau_I \circ \partial_H(t')) =$
$= \tau \cdot \partial_H \circ \tau_I(t')$ (induction hypothesis) $= \partial_H(\tau \cdot \tau_I(t')) = \partial_H(\tau_I(a) \cdot \tau_I(t')) = \partial_H \circ \tau_I(at') =$
$= \partial_H \circ \tau_I(t)$.
Case 4: if $t = t' + t''$, $\tau_I \circ \partial_H(t) = \tau_I \circ \partial_H(t' + t'') = \tau_I \circ \partial_H(t') + \tau_I \circ \partial_H(t'') = \partial_H \circ \tau_I(t') + \partial_H \circ \tau_I(t'')$ (induction hypothesis) $= \partial_H(\tau_I(t') + \tau_I(t'')) = \partial_H \circ \tau_I(t' + t'') = \partial_H \circ \tau_I(t)$.

4.10 Remark: Conditional axioms CA5-7 are special cases of more general properties of renaming operators, of which ∂_H and τ_I are two examples.
For more information about renaming operators, see Vaandrager [16].

5. Examples

5.1 Suppose we have two bags, linked together as in fig. 1.

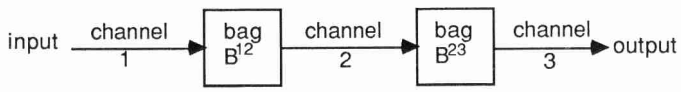

Fig. 1

We want to prove that the external behaviour of this system again is a bag. Therefore, we want to 'hide' the internal channel 2.

Let D be a finite set of data, and let the atomic actions ri(d), sj(d) be as in example 4.7. We define $c2(d) = r2(d) \mid s2(d)$, the **communication** of a $d \in D$ along channel 2. All other communications yield δ. Further, we define:

Encapsulation: $H = \{r2(d), s2(d) : d \in D\}$ (all unsuccessful communications).

Abstraction: $I = \{c2(d) : d \in D\}$ (all internal actions).

5.2 Theorem: $B^{13} = \tau_I \circ \partial_H(B^{12} \| B^{23})$.

Proof: $\partial_H(B^{12} \| B^{23}) = \sum_{d \in D} r1(d) \cdot \partial_H(s2(d) \| B^{12} \| B^{23})$

(using standard concurrency, see 2.11) =

$= \sum_{d \in D} r1(d) \cdot \partial_H \circ \partial_{\{s2(d)\}}(B^{12} \| (s2(d) \| B^{23}))$ (by CA5) =

$= \sum_{d \in D} r1(d) \cdot \partial_H \circ \partial_{\{s2(d)\}}(B^{12} \| \partial_{\{s2(d)\}}(s2(d) \| B^{23}))$ (by CA1, see note 1 below) =

$= \sum_{d \in D} r1(d) \cdot \partial_H(B^{12} \| (c2(d) \cdot s3(d) \| B^{23}))$ (CA5, and see note 2 below) =

$= \sum_{d \in D} r1(d) \cdot \partial_H(c2(d) \cdot s3(d) \| (B^{12} \| B^{23}))$ (standard concurrency) =

$= \sum_{d \in D} r1(d) \cdot \partial_H(c2(d) \cdot s3(d) \| \partial_H(B^{12} \| B^{23}))$ (by CA1, see note 3 below) =

$= \sum_{d \in D} r1(d) \cdot (c2(d) \cdot s3(d) \| \partial_H(B^{12} \| B^{23}))$ (by CA3, see note 4 below).

Using this result, we get that $\tau_I \circ \partial_H(B^{12} \| B^{23}) =$

$= \sum_{d \in D} r1(d) \cdot \tau_I(c2(d) \cdot s3(d) \| \partial_H(B^{12} \| B^{23})) =$

$= \sum_{d \in D} r1(d) \cdot \tau_I(\tau_I(c2(d) \cdot s3(d)) \| \tau_I \circ \partial_H(B^{12} \| B^{23}))$ (CA3 twice, use note 3) =

$= \sum_{d \in D} r1(d) \cdot \tau_I(\tau \cdot s3(d) \| \tau_I \circ \partial_H(B^{12} \| B^{23})) =$

$= \sum_{d \in D} r1(d) \cdot (\tau \cdot s3(d) \| \tau_I \circ \partial_H(B^{12} \| B^{23}))$ (by CA4, see note 5 below) =

$= \sum_{d \in D} r1(d) \cdot (s3(d) \| \tau_I \circ \partial_H(B^{12} \| B^{23}))$ (see note 6 below).

Therefore, the process $\tau_I \circ \partial_H(B^{12} \| B^{23})$ satisfies the defining equation of B^{13}. By the Recursive Specification Principle (see 2.9), we obtain $B^{13} = \tau_I \circ \partial_H(B^{12} \| B^{23})$.

Note 1: $\alpha(B^{12}) \mid (\alpha(s2(d)\|B^{23}) \cap \{s2(d)\} \subseteq \{r1(e), s2(e) : e\in D\} \mid \{s2(d)\} = \emptyset$ (each $d\in D$).
Note 2: Let $d\in D$. Then on the one hand
$c2(d)s3(d)\|B^{23} = c2(d)\cdot(s3(d)\|B^{23}) + \sum_{e\in D} r2(e)\cdot(s3(e)\|c2(d)s3(d)\|B^{23})$,

and on the other hand
$\partial_{\{s2(d)\}}(s2(d)\|B^{23}) = c2(d)\cdot\partial_{\{s2(d)\}}(s3(d)\|B^{23}) + \sum_{e\in D} r2(e)\cdot\partial_{\{s2(d)\}}(s3(e)\|s2(d)\|B^{23}) =$

$= c2(d)(s3(d)\|B^{23}) + \sum_{e\in D} r2(e)\cdot\partial_{\{s2(d)\}}(s3(e)\|s2(d)\|B^{23})$
(by CA3, see note 7 below) =

$= c2(d)(s3(d)\|B^{23}) + \sum_{e\in D} r2(e)\cdot\partial_{\{s2(d)\}}(s3(e)\|\partial_{\{s2(d)\}}(s2(d)\|B^{23}))$
(by CA1, argue as in note 1) =

$= c2(d)(s3(d)\|B^{23}) + \sum_{e\in D} r2(e)\cdot(s3(e)\|\partial_{\{s2(d)\}}(s2(d)\|B^{23}))$
(by CA3, see note 8 below) =

Thus we see that both process $c2(d)s3(d)\|B^{23}$ and process $\partial_{\{s2(d)\}}(s2(d)\|B^{23})$ satisfy the guarded equation

$$x = c2(d)\cdot(s3(d)\|B^{23}) + \sum_{e\in D} r2(e)\cdot(s3(e)\|x).$$

Therefore, by the Recursive Specification Principle,

$$c2(d)s3(d)\|B^{23} = \partial_{\{s2(d)\}}(s2(d)\|B^{23}).$$

Note 3: $\{c2(d), s3(d) : d\in D\} \mid A = \emptyset$, since r3(d) does not occur.
Note 4: $\alpha(c2(d)s3(d)\|\partial_H(B^{12}\|B^{23})) \cap H \subseteq$
$\subseteq [\{c2(d), s3(d)\} \cup (A\text{-}H) \cup \{c2(d), s3(d)\} \mid (A\text{-}H)] \cap H = (A\text{-}H) \cap H = \emptyset$, for each $d\in D$.
Note 5: $\alpha(\tau_I\circ\partial_H(B^{12}\|B^{23})\|\tau s3(d)) \cap I \subseteq [(A\text{-}I) \cup \{s3(d)\} \cup (A\text{-}I) \mid \{s3(d)\}] \cap I = [A\text{-}I] \cap I =$
$= \emptyset$, for each $d\in D$.
Note 6: $r1(d)(\tau s3(d)\|x) = r1(d)\tau s3(d)\|x = r1(d)s3(d)\|x = r1(d)(s3(d)\|x)$.
Note 7: $\alpha(s3(d)\|B^{23})) \cap \{s2(d)\} = [\{s3(d)\} \cup \{r2(e), s3(e) : e\in D\} \cup$
$\cup \{s3(d)\} \mid \{r2(e), s3(e) : e\in D\}] \cap \{s2(d)\} \subseteq [A\text{-}\{s2(d)\}] \cap \{s2(d)\} = \emptyset$, for each $d\in D$.
Note 8: $\alpha(s3(e)\|\partial_{\{s2(d)\}}(s2(d)\|B^{23})) \cap \{s2(d)\} \subseteq (\{s3(e)\} \cup (A\text{-}\{s2(d)\}) \cup \{s3(e)\} \mid A) \cap$
$\cap \{s2(d)\} \subseteq (A\text{-}\{s2(d)\}) \cap \{s2(d)\} = \emptyset$, for each $d,e\in D$.
This finishes the proof of theorem 5.2.

5.3 We can easily generalise the theorem above to the situation where we have more than 2 bags connected in a row. To illustrate, we will consider the case of 3 bags. We define ri(d), si(d), ci(d) as before (see 4.7, 5.1), and further:
Encapsulation: $Hn = \{sn(d), rn(d) : d\in D\}$ $(n=2,3)$; $H = H2 \cup H3$;
Abstraction: $In = \{cn(d) : d\in D\}$ $(n=2,3)$; $I = I2 \cup I3$.

5.4 Theorem: $B^{14} = \tau_I\circ\partial_H(B^{12}\|B^{23}\|B^{34})$.
Proof: By 5.2, we have $B^{24} = \tau_{I3}\circ\partial_{H3}(B^{23}\|B^{34})$ and $B^{14} = \tau_{I2}\circ\partial_{H2}(B^{12}\|B^{24})$.
Therefore $\tau_I\circ\partial_H(B^{12}\|B^{23}\|B^{34}) =$
$= \tau_{I2}\circ\tau_{I3}\circ\partial_{H2}\circ\partial_{H3}(B^{12}\|B^{23}\|B^{34})$ (CA5, CA6) =

$= \tau_{I2} \circ \partial_{H2} \circ \tau_{I3} \circ \partial_{H3}(B^{12} \| B^{23} \| B^{34})$ (CA7) =

$= \tau_{I2} \circ \partial_{H2} \circ \tau_{I3} \circ \partial_{H3}(B^{12} \| \partial_{H3}(B^{23} \| B^{34}))$ (CA1, see note 1 below) =

$= \tau_{I2} \circ \partial_{H2} \circ \tau_{I3}(B^{12} \| \partial_{H3}(B^{23} \| B^{34}))$ (CA3, see note 2 below) =

$= \tau_{I2} \circ \partial_{H2} \circ \tau_{I3}(B^{12} \| \tau_{I3} \circ \partial_{H3}(B^{23} \| B^{34}))$ (CA2, see note 3 below) =

$= \tau_{I2} \circ \partial_{H2} \circ \tau_{I3}(B^{12} \| B^{24})$ (by theorem 5.2) =

$= \tau_{I2} \circ \partial_{H2}(B^{12} \| B^{24})$ (CA4) =

$= B^{14}$ (by theorem 5.2).

To finish the proof, we only need to check some alphabets:

Note 1: $\alpha(B^{12}) \mid (\alpha(B^{23} \| B^{34}) \cap H3) \subseteq \{r1(d), s2(d) : d \in D\} \mid H3 = \emptyset$.

Note 2: $\alpha(B^{12} \| \partial_{H3}(B^{23} \| B^{34})) \cap H3) \subseteq [(A\text{-}H3) \cup (A\text{-}H3) \cup (A\text{-}H3) \mid (A\text{-}H3)] \cap H3 = \emptyset$.

Note 3: $\alpha(B^{12}) \mid \alpha(\partial_{H3}(B^{23} \| B^{34})) \cap I3) \subseteq A \mid I3 = \emptyset$.

5.5 Our final example is somewhat more involved. It considers a bag with test for empty. Such a bag (with input port 1 and output port 2) can be defined by the following guarded recursive specification (notations as before):

$$B\emptyset = \sum_{d \in D} (r1(d)B_d + s2(\emptyset)) \cdot B\emptyset$$

$$B_d = s2(d) + \sum_{e \in D} r1(e)(B_e \| B_d)$$

To see that this indeed defines a bag with test for empty, consider the following lemma.

5.6 Lemma: $\partial_{\{s2(\emptyset)\}}(B\emptyset) = B^{12}$ (notation from 4.7)

Proof: We prove the lemma with the Recursive Specification Principle, here applied to an *infinite* recursive specification. We will show that for all multisets G of elements of D we have

$$B^{12} \| (\|_{e \in G} s2(e)) = (\|_{e \in G} B_e) \cdot \partial_{\{s2(\emptyset)\}}(B\emptyset) \qquad (*)$$

(for $G=\emptyset$, we define $B^{12} \| (\|_{e \in \emptyset} s2(e)) = B^{12}$ and $(\|_{e \in \emptyset} B_e) \cdot \partial_{\{s2(\emptyset)\}}(B\emptyset) = \partial_{\{s2(\emptyset)\}}(B\emptyset)$),

by showing that both sides satisfy the same recursive specification.

We see that the lemma follows immediately from (*). To show (*), we consider two cases:

Case 1: $G=\emptyset$. Then $\partial_{\{s2(\emptyset)\}}(B\emptyset) = \sum_{d \in D} (r1(d) \cdot \partial_{\{s2(\emptyset)\}}(B_d) + \delta) \cdot \partial_{\{s2(\emptyset)\}}(B\emptyset) =$

$= \sum_{d \in D} r1(d) \cdot B_d \cdot \partial_{\{s2(\emptyset)\}}(B\emptyset)$ (by CA3, see note below),

and on the other hand $B^{12} = \sum_{d \in D} r1(d)(B^{12} \| s2(d))$ by definition.

Note: $\alpha(B_d) = \{s2(d)\} \cup \{r1(e) : e \in D\} \cup \cup\{\alpha(B_e \| B_d : e \in D\} = \{s2(d)\} \cup \{r1(e) : e \in D\} \cup \cup \cup\{\alpha(B_e : e \in D\} \cup \alpha(B_d) = \cup \cup\{\alpha(B_e \mid B_d : e \in D\}$. Since $\beta(B_d)$ is the least fixed point of this equation, it follows easily that $\beta(B_d) = \{s2(e), r1(e) : e \in D\}$. Thus $\beta(B_d) \cap \{s2(\emptyset)\} = \emptyset$, whence $\alpha(B_d) \cap \{s2(\emptyset)\} = \emptyset$.

Case 2: G≠Ø. In this case, we need the **Expansion Theorem** of Bergstra & Tucker [8]: let processes $x_1,...,x_n$ be given and assume the Handshaking Axiom (see 4.6). Then we can prove:

$$x_1\|x_2\|...\|x_n = \sum_{1\le i\le n} x_i\mathbb{L}(\|_{\substack{1\le k\le n\\ k\neq i}} x_k) + \sum_{1\le i<j\le n} (x_i|x_j)\mathbb{L}(\|_{\substack{1\le k\le n\\ k\neq i,j}} x_k).$$

The Expansion Theorem (ET) says that if we have a merge of a number of processes, then we can start with an action of one of the processes, or with a communication between two of them. Using the Expansion Theorem here, we get:

$$B^{12}\|(\|_{e\in G} s2(e)) = \sum_{d\in G} s2(d)(B^{12}\|(\|_{e\in G-\{d\}} s2(e))) + \sum_{d\in D} r1(d)(B^{12}\|(\|_{e\in G\cup\{d\}} s2(e))), \text{ and}$$

$$(\|_{e\in G} B_e)\cdot\partial_{\{s2(\emptyset)\}}(B\emptyset) = \sum_{d\in G} s2(d)(\|_{e\in G-\{d\}} B_e)\cdot\partial_{\{s2(\emptyset)\}}(B\emptyset) +$$

$$+ \sum_{d\in D} r1(d)(\|_{e\in G\cup\{d\}} B_e)\cdot\partial_{\{s2(\emptyset)\}}(B\emptyset).$$

5.7 Now suppose we want to link this bag with empty test to a regular bag. Between the two, we interpose a one-place buffer, that 'forgets' the empty test, i.e. a process defined by the following guarded recursive equation:

$$T\emptyset = \sum_{d\in D} (r2(d)s3(d) + r2(\emptyset))\cdot T\emptyset.$$

Thus we have the situation of fig. 2.

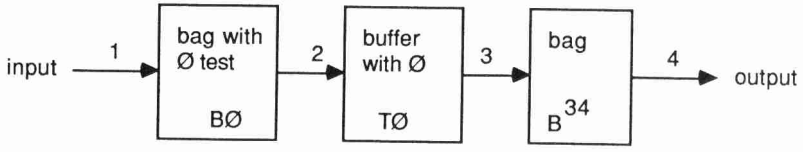

Fig. 2.

5.8 We define H2, H3, I2, I3 as in 5.3, and define in addition:

Encapsulation: $H\emptyset = \{r2(\emptyset), s2(\emptyset)\}$, $H = H2 \cup H3 \cup H\emptyset$;

Abstraction: $I = I2 \cup I3 \cup \{c2(\emptyset)\}$.

We want to prove the following theorem:

$$\tau_I\circ\partial_H(B\emptyset\|T\emptyset\|B^{34}) = \tau B^{14}.$$

Before we can prove this theorem, we need to prove a number of lemmas. First, define a (regular) one-place buffer T by:

$$T = \sum_{d\in D} r2(d)s3(d)\cdot T.$$

5.9 Lemma: $\partial_{\{r2(\emptyset)\}}(T\emptyset) = T$.

Proof: $\partial_{\{r2(\emptyset)\}}(T\emptyset) = \sum_{d\in D} (r2(d)s3(d) + \delta)\cdot\partial_{\{r2(\emptyset)\}}(T\emptyset) = \sum_{d\in D} r2(d)s3(d)\cdot\partial_{\{r2(\emptyset)\}}(T\emptyset)$.

Now use the Recursive Specification Principle.

5.10 Lemma: Define $P = \partial_{H2\cup H\emptyset}(B\emptyset\|T\emptyset)$. Then there is a process Q such that
$P = c2(\emptyset)\cdot P + Q\cdot P$ and $c2(\emptyset) \notin \alpha(Q)$.

Proof: We will give an infinite guarded recursive specification for Q. This specification has variables P(G) and P(G,d), for each multiset G of elements of D, and each $d\in D$. The intuitive meaning of these processes is that P(G) describes the bag with contents G, while the buffer is empty, and P(G,d) describes the bag with contents G, the buffer filled with d. We have the following equations:

Case 1: G=Ø. $Q = \sum_{d\in D} r1(d)\cdot P(\{d\})$

$P(\emptyset) = P$

$P(\emptyset,d) = s3(d) + \sum_{e\in D} r1(e)\cdot P(\{e\},d)$

Case 2: G≠Ø. $P(G) = \sum_{d\in G} c2(d)\cdot P(G-\{d\},d) + \sum_{d\in D} r1(d)\cdot P(G\cup\{d\})$

$P(G,d) = s3(d)\cdot P(G) + \sum_{e\in D} r1(e)\cdot P(G\cup\{e\},d)$

Then we have $P = \partial_{H2\cup H\emptyset}(B\emptyset\|T\emptyset) =$

$= c2(\emptyset)\cdot\partial_{H2\cup H\emptyset}(B\emptyset\|T\emptyset) + \sum_{d\in D} r1(d)\cdot\partial_{H2\cup H\emptyset}(B_d\cdot B\emptyset\|T\emptyset) =$

$= c2(\emptyset)\cdot P + \sum_{d\in D} r1(d)\cdot P(\{d\})\cdot P = c2(\emptyset)\cdot P + Q\cdot P,$

which follows from the following two equations:

(1) $\partial_{H2\cup H\emptyset}((\|_{e\in G} B_e)B\emptyset\|T\emptyset) = P(G)\cdot P$

(2) $\partial_{H2\cup H\emptyset}((\|_{e\in G} B_e)B\emptyset\|s3(d)T\emptyset) = P(G,d)\cdot P$

We prove these equations by showing that both sides satisfy the same guarded recursive specification.

Case 1: G=Ø. Equation (1) is shown above. To show (2):

$\partial_{H2\cup H\emptyset}(B\emptyset\|s3(d)T\emptyset) = s3(d)\cdot\partial_{H2\cup H\emptyset}(B\emptyset\|T\emptyset) + \sum_{e\in D}\partial_{H2\cup H\emptyset}(B_e\cdot B\emptyset\|s3(d)T\emptyset),$

and $P(\emptyset,d)\cdot P = (s3(d) + \sum_{e\in D} r1(e)\cdot P(\{e\},d))\cdot P = s3(d)\cdot P + \sum_{e\in D} r1(e)\cdot P(\{e\},d)\cdot P.$

Case 2: G≠Ø. For equation (1), see the following:

$\partial_{H2\cup H\emptyset}((\|_{e\in G} B_e)B\emptyset\|T\emptyset) = \sum_{e\in G} c2(d)\cdot\partial_{H2\cup H\emptyset}((\|_{e\in G-\{d\}} B_e)B\emptyset\|s3(d)T\emptyset) +$

$+ \sum_{d\in D} r1(d)\cdot\partial_{H2\cup H\emptyset}((\ \|_{e\in G\cup\{d\}} B_e)B\emptyset\|s3(d)T\emptyset)$ (by Expansion Theorem), and

$P(G)\cdot P = \sum_{d\in G} c2(d)P(G-\{d\},d)\cdot P + \sum_{d\in D} r1(d)P(G\cup\{d\})\cdot P.$

For equation (2), consider the following statements:

$\partial_{H2\cup H\emptyset}((\ \|_{e\in G} B_e)B\emptyset\|s3(d)T\emptyset) = s3(d)\cdot\partial_{H2\cup H\emptyset}((\ \|_{e\in G} B_e)B\emptyset\|T\emptyset) +$

$+ \sum_{f\in D} r1(f)\cdot\partial_{H2\cup H\emptyset}((\ \|_{e\in G\cup\{f\}} B_e)B\emptyset\|s3(d)T\emptyset)$ (by Expansion Theorem), and

$P(G,d)\cdot P = s3(d)P(G)\cdot P + \sum_{f\in D} r1(f)P(G\cup\{f\},d)\cdot P.$

This finishes the proof of the first half of the lemma. To prove the second half, it is not much trouble to show that for all multisets G and all $d\in D$,

$$\beta(P(G)) = \{r1(d), c2(d), s3(d) : d\in D\} \quad (G\neq\emptyset), \text{ and}$$
$$\beta(P(G,d)) = \{r1(d), c2(d), s3(d) : d\in D\}.$$

Thus, $\beta(Q) = \{r1(d), c2(d), s3(d) : d\in D\}$, whence $c2(\emptyset)\notin\beta(Q)$ and $c2(\emptyset)\notin\alpha(Q)$.

5.11 Note: If we can use *priorities* on atomic actions (see Baeten, Bergstra & Klop [2]), then Q can be defined by a finite recursive specification, as follows. If all actions s3(d) (for $d\in D$) have priority over all c2(e) (for $e\in D$), and θ implements this priority (i.e. $\theta(s3(d)x + c2(e)y) = s3(d)\theta(x)$), we can define

$$C_d = c2(d)T_d + \sum_{e\in D} r1(e)(C_e\|C_d) \quad \text{(for } d\in D)$$

$$T_d = s3(d) + \sum_{e\in D} r1(e)(T_d\|C_e) \quad \text{(for } d\in D)$$

$$Q = \sum_{d\in D} r1(d)\cdot\theta(C_d).$$

5.12 Lemma: $\tau_{\{c2(\emptyset)\}}(P) = \tau\cdot\partial_{\{c2(\emptyset)\}}(P)$.

Proof: By 5.10, we have $P = c2(\emptyset)P + QP$. Applying KFAR (see 3.4) to this equation, we get

$$\tau_{\{c2(\emptyset)\}}(P) = \tau\cdot\tau_{\{c2(\emptyset)\}}(QP) = \tau\cdot\tau_{\{c2(\emptyset)\}}(Q)\cdot\tau_{\{c2(\emptyset)\}}(P) =$$
$$= \tau\cdot Q\cdot\tau_{\{c2(\emptyset)\}}(P)$$

by CA4, since $c2(\emptyset)\notin\alpha(Q)$. On the other hand

$$\tau\cdot\partial_{\{c2(\emptyset)\}}(P) = \tau\cdot\partial_{\{c2(\emptyset)\}}(c2(\emptyset)P + QP) = \tau(\delta + \partial_{\{c2(\emptyset)\}}(QP)) =$$
$$= \tau\cdot\partial_{\{c2(\emptyset)\}}(Q)\cdot\partial_{\{c2(\emptyset)\}}(P) = \tau\cdot Q\cdot\partial_{\{c2(\emptyset)\}}(P) \text{ by CA3,}$$

so by the Recursive Specification Principle $\tau_{\{c2(\emptyset)\}}(P) = \tau\cdot\partial_{\{c2(\emptyset)\}}(P)$.

5.13 Lemma: $\tau_{I2\cup\{c2(\emptyset)\}}\circ\partial_{H2\cup H\emptyset}(B\emptyset\|T\emptyset) = \tau\cdot\tau_{I2}\circ\partial_{H2}(B^{12}\|T)$.

Proof: $\tau_{I2\cup\{c2(\emptyset)\}}\circ\partial_{H2\cup H\emptyset}(B\emptyset\|T\emptyset) =$

$= \tau_{I2}\circ\tau_{\{c2(\emptyset)\}}\circ\partial_{H2\cup H\emptyset}(B\emptyset\|T\emptyset)$ (by CA6) =

$= \tau_{I2}(\tau\cdot\partial_{\{c2(\emptyset)\}}\circ\partial_{H2\cup H\emptyset}(B\emptyset\|T\emptyset)$ (by 5.12) =

$= \tau\cdot\tau_{I2}{}^{\circ}\partial_{H2}{}^{\circ}\partial_{\{c2(\emptyset)\}\cup H\emptyset}(B\emptyset\|T\emptyset)$ (by CA5) =

$= \tau\cdot\tau_{I2}{}^{\circ}\partial_{H2}{}^{\circ}\partial_{\{c2(\emptyset)\}\cup H\emptyset}(\partial_{\{c2(\emptyset)\}\cup H\emptyset}(B\emptyset)\|\partial_{\{c2(\emptyset)\}\cup H\emptyset}(T\emptyset))$
(by CA1, see note 1 below) =

$= \tau\cdot\tau_{I2}{}^{\circ}\partial_{H2}{}^{\circ}\partial_{\{c2(\emptyset)\}\cup H\emptyset}(\partial_{\{c2(\emptyset),\, r2(\emptyset)\}}{}^{\circ}\partial_{\{s2(\emptyset)\}}(B\emptyset)\|\partial_{\{c2(\emptyset),\, s2(\emptyset)\}}{}^{\circ}\partial_{\{r2(\emptyset)\}}(T\emptyset))$
(by CA5) =

$= \tau\cdot\tau_{I2}{}^{\circ}\partial_{H2}{}^{\circ}\partial_{\{c2(\emptyset)\}\cup H\emptyset}(\partial_{\{c2(\emptyset),\, r2(\emptyset)\}}(B^{12})\|\partial_{\{c2(\emptyset),\, s2(\emptyset)\}}(T))$
(by 5.6 and 5.9) =

$= \tau\cdot\tau_{I2}{}^{\circ}\partial_{H2}{}^{\circ}\partial_{\{c2(\emptyset)\}\cup H\emptyset}(B^{12}\|T)$ (by CA3, see note 2 below) =

$= \tau\cdot\tau_{I2}{}^{\circ}\partial_{H2}(B^{12}\|T)$ (by CA3, see note 3 below).

Note 1: This is because $\{c2(\emptyset)\}\cup H\emptyset$ is closed under communication.

Note 2: $\alpha(B^{12}) = \{r1(d), s2(d) : d\in D\}$ by 4.7, and $\alpha(T) = \{r2(d), s3(d) : d\in D\}$ is easily proved.

Note 3: $\alpha(B^{12}\|T) = \alpha(B^{12}) \cup \alpha(T) \cup \alpha(B^{12}) \,|\, \alpha(T) = \{r1(d), s2(d), c2(d), r2(d), s3(d) : d\in D\}$.

5.14 Lemma: $\tau_{I3}{}^{\circ}\partial_{H3}(T\|B^{34}) = B^{24}$.

Proof: $\tau_{I3}{}^{\circ}\partial_{H3}(T\|B^{34}) = \tau_{I3}(\sum_{d\in D} r2(d)\cdot\partial_{H3}(s3(d)T\|B^{34}) =$

$= \sum_{d\in D} r2(d)\cdot\tau_{I3}(c3(d)\cdot\partial_{H3}(T\|B^{34}\|s4(d)) =$

$= \sum_{d\in D} r2(d)\cdot\tau\cdot\tau_{I3}{}^{\circ}\partial_{H3}(\partial_{H3}(T\|B^{34})\|s4(d))$ (by CA1, note that $A\,|\,\{s4(d)\} = \emptyset$) =

$= \sum_{d\in D} r2(d)\cdot\tau_{I3}(\partial_{H3}(T\|B^{34})\|s4(d))$ (by CA3) =

$= \sum_{d\in D} r2(d)\cdot\tau_{I3}(\tau_{I3}{}^{\circ}\partial_{H3}(T\|B^{34})\|s4(d))$ (by CA2) =

$= \sum_{d\in D} r2(d)\cdot(\tau_{I3}{}^{\circ}\partial_{H3}(T\|B^{34})\|s4(d))$ (by CA4).

Therefore, $\tau_{I3}{}^{\circ}\partial_{H3}(T\|B^{34})$ satisfies the defining equation of B^{24}, so by the Recursive Specification Principle $\tau_{I3}{}^{\circ}\partial_{H3}(T\|B^{34}) = B^{24}$.

5.15 Lemma: (Van Glabbeek [10]) $ACP_\tau \vdash \tau x\|y = \tau(x\|y)$.

Proof: $\tau x\|y = \tau x\,\llcorner\, y + y\,\llcorner\,\tau x + \tau x\,|\,y = \tau(x\|y) + y\,\llcorner\,\tau x + \tau x\,|\,y$. Therefore $\tau x\|y = \tau x\|y + \tau(x\|y)$. On the other hand $\tau(x\|y) = \tau x\,\llcorner\, y = \tau\tau x\,\llcorner\, y = \tau(\tau x\|y) = \tau(\tau x\|y) + \tau x\|y$. Therefore $\tau(x\|y) = \tau(x\|y) + \tau x\|y$.

5.16 Now we can finally prove the theorem refered to in 5.8.

Theorem: $\tau_I{}^{\circ}\partial_H(B\emptyset\|T\emptyset\|B^{34}) = \tau B^{14}$.

Proof: $\tau_I{}^{\circ}\partial_H(B\emptyset\|T\emptyset\|B^{34}) =$

$= \tau_{I3}{}^{\circ}\tau_{I2\cup\{c2(\emptyset)\}}{}^{\circ}\partial_{H3}{}^{\circ}\partial_{H2\cup H\emptyset}(B\emptyset\|T\emptyset\|B^{34})$ (by CA5 and CA6) =

$= \tau_{I3}{}^{\circ}\partial_{H3}{}^{\circ}\tau_{I2\cup\{c2(\emptyset)\}}{}^{\circ}\partial_{H2\cup H\emptyset}(B\emptyset\|T\emptyset\|B^{34})$ (by CA7) =

$= \tau_{I3} \circ \partial_{H3} \circ \tau_{I2 \cup \{c2(\emptyset)\}} \circ \partial_{H2 \cup H\emptyset}(\partial_{H2 \cup H\emptyset}(B\emptyset \| T\emptyset) \| B^{34})$
(by CA1, see note 1 below) =
$= \tau_{I3} \circ \partial_{H3} \circ \tau_{I2 \cup \{c2(\emptyset)\}}(\partial_{H2 \cup H\emptyset}(B\emptyset \| T\emptyset) \| B^{34})$ (by CA3, see note 1 below) =
$= \tau_{I3} \circ \partial_{H3} \circ \tau_{I2 \cup \{c2(\emptyset)\}}(\tau_{I2 \cup \{c2(\emptyset)\}} \circ \partial_{H2 \cup H\emptyset}(B\emptyset \| T\emptyset) \| B^{34})$
(by CA2, see note 1 below) =
$= \tau_{I3} \circ \partial_{H3}(\tau_{I2 \cup \{c2(\emptyset)\}} \circ \partial_{H2 \cup H\emptyset}(B\emptyset \| T\emptyset) \| B^{34})$ (by CA4, see note 1 below) =
$= \tau_{I3} \circ \partial_{H3}(\tau \cdot \tau_{I2} \circ \partial_{H2}(B^{12} \| T) \| B^{34})$ (by 5.13) =
$= \tau \cdot \tau_{I3} \circ \partial_{H3}(\tau_{I2} \circ \partial_{H2}(B^{12} \| T) \| B^{34})$ (by 5.15) =
$= \tau \cdot \tau_{I3} \circ \partial_{H3} \circ \tau_{I2} \circ \partial_{H2}(\tau_{I2} \circ \partial_{H2}(B^{12} \| T) \| B^{34})$ (by CA3, CA4, see note 2 below) =
$= \tau \cdot \tau_{I3} \circ \partial_{H3} \circ \tau_{I2} \circ \partial_{H2}(B^{12} \| T \| B^{34})$ (by CA1, CA2, see note 3 below) =
$= \tau \cdot \tau_{I2} \circ \partial_{H2} \circ \tau_{I3} \circ \partial_{H3}(B^{12} \| T \| B^{34})$ (by CA7) =
$= \tau \cdot \tau_{I2} \circ \partial_{H2} \circ \tau_{I3} \circ \partial_{H3}(B^{12} \| \tau_{I3} \circ \partial_{H3}(T \| B^{34}))$ (by CA1, CA2, see note 4 below) =
$= \tau \cdot \tau_{I2} \circ \partial_{H2}(B^{12} \| \tau_{I3} \circ \partial_{H3}(T \| B^{34}))$ (by CA3, CA4, see note 4 below) =
$= \tau \cdot \tau_{I2} \circ \partial_{H2}(B^{12} \| B^{24})$ (by 5.14) =
$= \tau \cdot B^{14}$ (by 5.2).

Note 1: By 4.7, $\alpha(B^{34}) = \{r3(d), s4(d) : d \in D\}$. Thus $\alpha(B^{34}) \mid (A - \{s3(d) : d \in D\}) = \emptyset$, and also $((A - (H2 \cup H\emptyset)) \cup \alpha(B^{34})) \cap (H2 \cup H\emptyset) = \emptyset$, and $((A - (I2 \cup \{c2(\emptyset)\})) \cup \alpha(B^{34})) \cap (I2 \cup \{c2(\emptyset)\}) = \emptyset$.

Note 2: Likewise, we show $((A-H2) \cup \alpha(B^{34})) \cap H2 = \emptyset$ and $((A-I2) \cup \alpha(B^{34})) \cap I2 = \emptyset$.

Note 3: $\alpha(B^{12} \| T) \mid \alpha(B^{34}) =$
$= (\{r1(d), s2(d) : d \in D\} \cup \{r2(d), s3(d) : d \in D\} \cup \{c2(d) : d \in D\}) \mid \{r3(d), s4(d) : d \in D\} =$
$= \{c3(d) : d \in D\}$. The result is $\emptyset$ if we intersect $\alpha(B^{12} \| T)$ with I2 or H2.

Note 4: $\alpha(B^{12}) \mid \alpha(T \| B^{34}) = \{r1(d), s2(d) : d \in D\} \mid \{r2(d), s3(d), c3(d), r3(d), s4(d) : d \in D\}$
$= \{c2(d) : d \in D\}$. The result is $\emptyset$ if we intersect $\alpha(T \| B^{34})$ with I3 or H3.

References.

[1] J.C.M.Baeten, J.A.Bergstra & J.W.Klop, *Conditional axioms and α/β-calculus in process algebra,* report CS-R8502, Centre for Mathematics and Computer Science, Amsterdam 1985.

[2] J.C.M.Baeten, J.A.Bergstra & J.W.Klop, *Syntax and defining equations for an interrupt mechanism in process algebra,* Fund. Inf. IX (2), pp. 127-168, 1986.

[3] J.C.M.Baeten, J.A.Bergstra & J.W.Klop, *On the consistency of Koomen's Fair Abstraction Rule,* report CS-R8511, Centre for Mathematics and Computer Science, Amsterdam 1985, to appear in Theor. Comp. Sci.

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

[5] J.A.Bergstra & J.W.Klop, *The algebra of recursively defined processes and the algebra of regular processes,* Proc. 11th ICALP, Antwerpen (ed. J.Paredaens), Springer LNCS 172, pp. 82-95, 1984.

[6] J.A.Bergstra & J.W.Klop, *Process algebra for synchronous communication,* Information & Control 60 (1/3), pp. 109-137, 1984.
[7] J.A.Bergstra & J.W.Klop, *Algebra of communicating processes with abstraction,* Theor. Comp. Sci. 37, pp. 77-121, 1985.
[8] J.A.Bergstra & J.V.Tucker, *Top-down design and the algebra of communicating processes,* Sci. of Comp. Progr. 5 (2), pp. 171-199, 1984.
[9] S.Brookes, C.Hoare & W.Roscoe, *A theory of communicating sequential processes,* JACM 31 (3), pp. 560-599, 1984.
[10] R.J. van Glabbeek, personal communication, 1986.
[11] M.Hennessy & G.Plotkin, *A term model for CCS,* Proc. 9th MFCS, Poland (1980), Springer LNCS 88.
[12] C.A.R.Hoare, *Communicating Sequential Processes,* Prentice Hall 1985.
[13] R.Milner, *A calculus of communicating systems,* Springer LNCS 92, 1980.
[14] E.-R.Olderog, *Specification oriented programming,* to appear.
[15] J.Sifakis, *Property preserving homomorphisms of transition systems,* Proc. Logics of Programs (1983), Springer LNCS 164, 1984.
[16] F.W.Vaandrager, *Verification of two communication protocols by means of process algebra,* report CS-R8608, Centre for Mathematics and Computer Science, Amsterdam 1986.

QUESTIONS AND ANSWERS

Olderog: How do you get axioms for your system without having a particular model?

Baeten: We analyse the axioms by comparing different models such as the initial algebra and the process graph model.

Best: You also need another trick: the 'recursive specification principle'!

Baeten: This principle means that every guarded recursive specification has a unique solution. This is an assumption in our approach and non-trivial to prove, for particular algebras. But since I am not talking about one model, I am just assuming it and have only to show that it is consistent. If you deal with models, of course, you have to prove it.

Jouannaud: Did you try to automate these proofs, and in particular the system verifications?

Baeten: No, I believe the proof we have presented is not automatizable. I cannot say these proofs

can be automatized, this may be possible in the future, I don't know. Maybe then there will be some large system verifications where automatized proofs could be useful. But we are not at a point that we can say we can do such things.

de Nicola: Are there any problems with the conditional axioms of the α/β - calculus for alphabets, since you say the alphabet is undecidable?

Baeten: No, you can work with approximations.

de Nicola: Do you know a method of deciding which subcalculi of your calculus can be completely and effectively axiomatized?

Baeten: I don't know. This is still in the future. We are all looking at such small examples, but it becomes important, if we get large systems. We do not have enough experience to really say something about that.

de Nicola: If one starts by writing axioms instead of starting from an operational model, one should start with axioms which can be put on the machine. I see two ways of using the axioms: either to justify the operational model or to start with some axioms which make definitely sense and then to put them on the machine and to see what their consequences are.

Baeten: I think, the use of the axioms is to be able to prove algebraically whether two processes are equal in order to understand the meaning of processes.

de Nicola: I agree.

Hehner: You wanted to have an effective approximation to the alphabet based on projections. The nth projection was defined informally and I am wondering, if that can be made effective. Can you at least give me the intuition for the effective calculation of the nth projection?

Baeten: You start from the given guarded recursive specification. And this means that every variable is preceeded by an atom.

Hehner: Yes, o.k., in the case of guarded definitions I can understand that. The projection does not apply then when it is not guarded?

Baeten: Well, every process can be given in three ways: either as a closed term or as a solution of a guarded recursive specification or as an abstraction from such a process. Only the third case is probably nontrivial with respect to projections.