BMACP

J.A. Bergstra
Programming Research Group, University of Amsterdam
Department of Philosophy, University of Utrecht

J.W. Klop
Department of Software Technology, Centre for Mathematics and Computer Science
Department of Computer Science, Free University Amsterdam

ABSTRACT. A new axiom system BMACP combines the axiom systems BMA (basic module algebra) and ACP (algebra of communicating processes) in such a way that the addition operator (+) which stands for module combination in BMA and for alternative composition of processes in ACP are both realised by the same operator (+) in BMACP. A closed process-module expression in BMACP denotes a process together with a specification of some auxiliary processes and all of these may be parametrised by some other processes.

This paper was written in honour of J.W. de Bakker on the occasion of the 25th anniversary of his association with the CWI. This work has been sponsored in part by ESPRIT project 432 METEOR.

1. MIXING PROCESS EXPRESSIONS AND DECLARATIONS

A topic that currently receives some interest is the design of specification languages in which process algebra in some form is used to describe dynamic aspects of target systems. Several difficulties have to be mastered mainly due to the fact that these specification languages combine declarative parts (declarations of data types, action alphabets, process names, state operators) and imperative parts (process expressions):

(i) How to syntactically combine process descriptions and data type descriptions? (LOTOS [Br 88] provides a sense of direction, PSF_d [M&V 88] redesigns a part of LOTOS in terms of ACP [BK 84] and ASF [BHK 89], further CRL [SP 89] a language that is being defined in the RACE project SPECS is yet another combination.)

(ii) How to define the meaning of modular algebraic specifications involving process descriptions (noticing the fact that for infinite processes bisimulation semantics is not obtained as an initial algebra semantics of their finite equational process specifications in process algebra, this in contrast with the case for finite processes)?

(iii) How to design formalisms combining process and data descriptions in such a way that the term rewriting paradigm is exploited best for both data and process specifications?

(iv) To what extent is equational logic sufficiently expressive for specification languages that combine data and process definitions?

(v) What are appropriate scope rules for local declarations of actions, process names and data?

(vi) What, if any, is the relationship between encapsulation and abstraction in process algebra and data abstraction (information hiding) as it occurs in static data type specifications.

2. MODULE ALGEBRA, A WAY OUT FOR THE LAZY THEORIST

The main dogma of module algebra is that the modularization constructions for specification languages fail to have a standard semantics and even fail to possess a simple model that reflects the intuition of the naive practitioner in the way the trace model describes the behavior of sequential and concurrent systems. Thus in [BHK 86] various models for module algebra have been found which are all quite different. In fact it is rather the absence of a single convincing model than the presence of various nice models which is the message of [BHK 86].

Modular specifications of processes can be described by embedding them in a module algebra on top of the process algebra. For this to be useful there is no need whatsoever for a 'nice' compositional semantics of the module algebra that assigns to modular process specifications a meaning that is intrinsic in terms of processes, behaviors, transition systems or something similar. Module algebra suggests the following minimalistic approach to the semantic problems connected with modularized specification languages.

(i) Introduce an algebraic syntax that allows to describe the module expressions that may be relevant for a particular (fragment of) a specification language that is being studied. Denote the resulting module expressions with ME. Define which expressions are normal forms, thus defining a class NMF(ME). The normal forms should show little or none of the modularisation mechanisms.

(ii) Provide a calculus C (or rather an equational proof system) that allows to transform each module expression to a normal form in the sense of (i).

(iii) Define a meaning for the module expressions in normal form in terms of the underlying concepts. It may be the case that only a part NMF* of NMF(ME) can be provided with a proper meaning. NMF* contains the 'nice' normal forms. The meaning of B in NMF* is denoted with SEM(B). It may well be rather difficult to pin down exactly the nice cases NMF*.

(iv) Prove that this meaning for NMF* is independent from the way in which the normal form was obtained by equational reasoning in the calculus C. Thus if A in ME has two normal forms B1 and B2 then B1 is part of NMF* if and only B2 is and in that case SEM(B1) = SEM(B2). This fact will then be referred to as the soundness of the calculus C. (One may consider the fact that C admits transformation to normal forms as completeness.)

The virtue of this procedure is that the meaning of a complex module expression A can be understood as follows:

(i) If A is an entire system then find a normal form B for A. If B is in NMF* then SEM(B) is the semantics of A.

(ii) If A is a component that has to be put in a context C[.] then one may safely simplify A using the rules of the calculus without the risk of changing SEM(C[A]) if it is defined. In particular A can be reduced to a normal form B and its intuitive meaning is then λ C[.]. C[B].

For the semantic description of complex specification languages the above program is much more easily carried out than the definition of an abstract and compositional semantics that provides a nice and natural mathematical meaning to every expression in ME. Of course this simply means that one (temporarily) withdraws from mathematical semantics and focusses on term models.

2.1. Fixed points and the μ-construct

We introduce an extension BMACP of ACP from [BK 84] by adding to it a part of the module algebra BMA of [BHK 86]. This extension should help in the solution of the problems (i), (ii) and (v) mentioned in 1 above. Further the axiom system BMACP sheds light on the status of the μ-operator in process algebra. We expand on the latter point first. The authors started working in process algebra after a talk by De Bakker held in Utrecht in June 1982. Then he posed the problem about the existence of fixed points of unguarded recursion equations in the topological model of processes that he developed in cooperation with Zucker (see [deBZ 82]). Our approach as reported in [BK 82] was to restrict attention to the case with a finite set of atomic actions and to define the equational axiom system PA involving alternative composition, sequential composition, merge and an auxiliary function named left merge. This axiom system PA has an initial algebra A_ω. This model allows finite projections A_n in which all processes are cut off after n steps. These finite projections are also models of PA. Then it was shown that the family of algebras A_n has a projective limit denoted with A_∞ which is a model of PA and more importantly can be isomorphically embedded in the aforementioned model of De Bakker and Zucker. By proving that all recursion equations can be solved in all finite projection algebras A_n it follows immediately that all equations are solvable in A_∞ and the problem posed by De Bakker was solved.

Now one easily proves that an unguarded equation of the form $X = P(X)$ has more then one solution in A_∞. In our view the algebra A_∞ stands out as a typical example of a process algebra. We have not yet found a natural way, however, to single out one canonical solution of (unguarded) recursion equations in A_∞. It follows that the definition of a μ-construct in the style of CCS [M 80] poses difficulties in A_∞. Indeed $\mu X . P(X)$ must denote some 'fixed' solution of the equation $X = P(X)$. Exactly because of these problems the first author strongly feels that the μ-construct should not be a part of process algebra. (Due to his history in lambda calculus the second author has a genuine sympathy for the binding mechanism of the μ-construct, however.) The axiom

system BMACP that we will describe allows the use of fixed point definitions within process expressions without any presuppositions concerning the (canonical) solvability of equations in a model.

It should be added immediately that the semantic pretentions of BMACP are minimal. Thus there is no implication that we can assign a meaning to fixed point constructions more often or in any better way than is possible with other theories. Rather conversely the meaning of the fixed point constructions is left completely open.

It should be noticed that the process theory of De Bakker and Zucker combines aspects of earlier process theories such as CSP [H 78] and CCS [M 80] by simultaneously using general sequential composition (taken from CSP) and a branching time semantics (taken from CCS). The main difference between CCS and the axiom system ACP from [BK 84] is due to the fact that in ACP general sequential composition is a primitive operator and not just prefix multiplication. In this sense ACP is an indeed an algebraic form of the model proposed in [deBZ 82].

This paper is not the first one that combines process algebra and module algebra. In [vGV 88] a much more natural combination is used. There the export mechanism of module algebra is used to hide the auxiliary functions left merge and communication merge after they have been helpful for the specification of the merge. This is needed if a transition to more abstract semantic models is to be made by adding equations that may be inconsistent in the presence of these auxiliary operators but are consistent in their absence.

2.2. Processes involving declarations

We can imagine that readers find our proposed BMACP rather grotesque and perhaps even clumsy. One of our purposes with the development of ACP is to provide extensions of ACP which are helpful for the design of specification languages. It turns out that each particular specification language design project leads to a multitude of matters of marginal scientific interest about which so-called 'design decisions' have to be made. The extension of ACP to BMACP addresses one of these marginal and somewhat unattractive aspects: scope rules for local process declarations in modular process specifications. The problem involved in these scope rules is easily illustrated with the main equation about δ in ACP:

$$\delta \cdot X = \delta$$

If X contains a process declaration with an unbounded scope then the equation is simply false. In BMACP we replace this equation by

$$\delta \cdot X = \partial(X).$$

Here A is the collection of all atomic actions and ∂ (process removal) replaces all atoms in X on an active position by δ. Now ∂ leaves the declarations of X unaffected, in fact it computes the declarations of X. ACP can be retrieved from BMACP by assuming the absence of declarations in

process expressions (i.e. $\partial(X) = \delta$). (Notice that the axiom system PA is obtained in a similar way from ACP. If all communications lead to δ then ACP reduces to PA_δ. So this way of extending a process algebra axiomatisation has precedents, be it that BMACP seems to have no models that can be viewed as a process algebra in any useful sense.)

2.3. A module algebra of recursive process specifications

Similar to the module algebra of [BKH 86] we may view individual recursion equations as atomic modules and import them to a class of modules (or rather module expressions) by an embedding operator $\langle\rangle$. Modules are then composed by means of an infix operator + which is then overloaded to denote the combination of specification modules as well as alternative composition of processes. Thus a list of defining equations will be represented as a combination of its elements (put between angular brackets). An example is as follows:

$\langle P = a \cdot Q\rangle + \langle Q = b \cdot Q\rangle$.

Now in many cases one is interested in process expressions that are written with use of the operators of process algebra and names of recursively defined processes. We propose that such process expressions may occur in sums besides recursion equations between angular brackets. From a different viewpoint one may say that we extend the algebra of process expressions with expressions of the form $\langle P = X\rangle$ with P a process name and X a process expression. A typical example is:

$a \cdot Q + \langle Q = b \cdot Q\rangle$.

If two such process expressions are combined with + their process parts are added in the way of process algebra and their declaration parts are added in the way of module algebra. If two such processes are merged then their process parts are merged and again the declaration parts are combined in the way of process algebra. The process part of a pure declaration like $\langle R = b \cdot R\rangle$ is just δ.

It will turn out that BMA contains an erasing axiom $X = X + (V \,\square\, X)$ which will be replaced by $X = X + \partial(V \,\square\, X)$. Notice that for purely declarative module expressions the identity $\partial(X) = X$ holds.

3. BMACP(A,γ,N), THE SIGNATURE

Let A be a finite collection of atomic actions not including δ. A_δ denotes the union of A and $\{\delta\}$. N is an infinite alphabet of process names disjoint from A. The function γ denotes the communication function which is a partial mapping of type $A \times A \to A$ which is associative and commutative. We will first describe the signature of the axiom system BMACP(A,γ,N). The role of the parameters A and N is to provide collections of constants for the sorts AP(atomic processes)

and PN (process names). The most important sort (type of interest) is the sort PM of process modules which at the same time plays the role of the processes of ACP and of the modules in BMA. The sort SIG wil contain signatures of process modules. In our setting such a signature is nothing more than the finite set of process names that are declared in the module.

begin signature $\Sigma_{BMACP}(A, N)$

sorts	PN	(process names)
	AA	(atomic actions)
	AP	(atomic processes)
	PM	(process modules)
	ER	(elementary renamings = pairs of process names)
	SIG	(signatures = finite subsets of PN)
constants		
	$a \rightarrow AA$	(for $a \in A$, atomic action)
	$\delta \rightarrow AP$	(deadlock)
	$\emptyset \rightarrow SIG$	(empty signature)
	$n \rightarrow PN$	(for $n \in N$, process name)
	$Id \rightarrow ER$	(identity)
functions (canonical embeddings)		
	$i\_aa\_ap: AA \rightarrow AP$	(embedding of actions into atomic processes)
	$i\_ap\_pm: AP \rightarrow PM$	(embedding of atomic processes in PM)
	$i\_pn\_sig: PN \rightarrow SIG$	(embedding of process names in SIG)
	$i\_pn\_pm: PN \rightarrow PM$	(embedding of process names in PM)
functions (inherited from process algebra)		
	$+: PM \times PM \rightarrow PM$	(alternative composition, serving as module combination as well)
	$\cdot: PM \times PM \rightarrow PM$	(sequential composition)
	$\|\|: PM \times PM \rightarrow PM$	(merge)
	$\lfloor\!\lfloor: PM \times PM \rightarrow PM$	(left merge)
	$\|: PM \times PM \rightarrow PM$	(communication merge)
	$\partial_H: PM \rightarrow PM$	(encapsulation, for each subset H of A)
functions (inherited from module algebra)		
	$+: SIG \times SIG \rightarrow SIG$	(combination of signatures)
	$\cap: SIG \times SIG \rightarrow SIG$	(intersection of signatures)
	$\Sigma: PM \rightarrow SIG$	(visible signature)
	$T: SIG \rightarrow PM$	(embedding of signatures in PM)
	$\square: SIG \times PM \rightarrow PM$	(export)
	$r: PN \times PN \rightarrow ER$	(elementary renaming / permutation)
	$\Sigma: ER \rightarrow SIG$	(signature implicit in renaming)
	$\cdot: ER \times SIG \rightarrow SIG$	(application of renaming)
	$\cdot: ER \times PM \rightarrow PM$	(application of renaming)
	$\cdot: ER \times PN \rightarrow PN$	(application of renaming)
functions (new)		
	$\partial: PM \rightarrow PM$	(process removal)
	$< = >: PN \times PM \rightarrow PM$	(process declaration module)

end signature $\Sigma_{BMACP}(A, N)$.

4. (META)VARIABLES, IMPLICIT EMBEDDING CONVENTION, AXIOM SCHEMES

4.1. Declaration of variables

The axioms of BMACP(A,γ,N) are introduced in the following section 5. There are some preliminaries to the listing of the axioms that need detailed attention. First we will declare the variables that will be used in the axioms:

```
begin variables for Σ_BMACP(A, N)
     P0, Q0, R0 → PN        [metavariables over N]
     P, Q, R → PN
     X, Y, Z → PM
     U, V, W → SIG
     a0, b0, → AA           [metavariables over A]
     a, b, c → AP
     r → ER
end variables
```

We will discuss the meaning of the attribute metavariable below. First we need an explanation of the (partial) implicit embedding convention.

4.2. Implicit embedding convention, implicit abbreviation convention, generalised implicit embedding and abbreviation convention

The *implicit embedding convention* allows to omit the embedding functions i_aa_ap, i_ap_pm, i_pn_sig, i_pn_pm whenever this will not lead to ambiguities. This means that equations and terms will be written in such a way that it is always clear how to augment an expression with the embedding functions in order to obtain a well-typed term or equation. Thus a precondition for an expression, equation or conditional equation to be well-formed is that it allows a unique type assignment involving the introduction of embeddings. Here it is understood that a variable must always be declared with a unique type in advance.

The *implicit abbreviation convention* allows to use prefixes of the names of the embeddings if these contain enough information to obtain unambiguous typing. Thus besides i_aa_ap, i_ap_pm, i_pn_sig and i_pn_pm, also i, i_aa, i_ap and i_pn are admitted. The convention will the allow to complete the name of the embedding function such as to obtain a term that is well-typed. Again there must be a unique way to do this. Moreover it is allowed to add additional embeddings if that helps to obtain a well-formed expression or equation again under the condition that this can be done in a unique way only.

A peculiar fact is that the implicit embedding convention may weaken the language because some identities can only be written in a way that allows additional but unintended typings. By prefixing the equation with [¬IE] it is indicated that no implicit embeddings and abbreviations are used or allowed. This introduces a complication for the syntax which is completely harmless for

the human user. We provide two examples: (i) On basis of the signature $\Sigma_{BMACP}(A, N)$ one obtains the PM expression

$$T(i\_pn\_sig(P) + \Sigma(X)) + i\_ap\_pm(a) \cdot \delta + i\_pn\_pm(Q) \; \square$$
$$(i\_pn\_pm(P) \parallel \langle P = i\_aa\_pm(b) \cdot i\_pn\_pm(P) + i\_aa\_pn(c) \rangle).$$

Using the implicit abbreviation convention this expression is written as:

$$T(i(P) + \Sigma(X)) + i(a) \cdot \delta + i(Q) \; \square \; (i(P) \parallel \langle P = i(b) \cdot i(P) + i(c) \rangle).$$

Then using the implicit embedding convention this expression is written as:

$$T(P + \Sigma(X)) + a \cdot \delta + Q \; \square \; (P \parallel \langle P = b \cdot P + c \rangle).$$

(ii) The equation $Id \cdot P = P$ is not well-formed because it has three different correct completions:

$Id \cdot P = P$, $Id \cdot i\_pn\_sig(P) = i\_pn\_sig(P)$ and

$Id \cdot i\_pn\_pm(P) = i\_pn\_pm(P)$.

Finally the *generalised implicit embedding and abbreviation convention* allows to use expressions having different completions to a well-typed expression if on basis of the other axioms the different completions can be proved equal. This convention should be used with care. A typical example is the expression $(r \cdot P) \parallel Q$. Obviously the subexpression $r \cdot P$ must have type PM. But that can be done using different completions: $r \cdot i\_pn\_pm(P)$ and $i\_pn\_pm(r \cdot P)$. Based on the presence of an axiom that identifies both expressions it is legitimate to use $r \cdot P$ as an expression of type PM.

It should be noticed that the mentioned conventions serve only one purpose: to allow a readable presentation of an algebraic specification in a many-sorted language. At the level of abstract syntax one always means the fully disambiguated version of the text.

4.3. Axiom schemes with metavariables

The axioms involving variables over AA and PN should in fact be understood as axiom schemes where every possible substitution from A resp. N is generated. Therefore negative conditions such as $P \neq Q$ are allowed and must be understood as restrictions on the number of substitutions that are required. Thus with $A = \{a1, a2, .., ak\}$ and $N = \{p1, p2, ..\}$ axioms should be read as follows:

$a \mid \delta = \delta$ stands for:

$$i\_ap\_pn(a) \mid i\_ap\_pn(\delta) = i\_ap\_pn(\delta)$$

$a0 \mid b0 = \gamma(a0, b0)$ if $G(a0, b0)$ is defined stands for

$$\{i\_ap\_pn(an) \mid i\_ap\_pn(am) = i\_ap\_pn(\gamma(an, am) \mid 1 \leq n \leq k, 1 \leq m \leq k, \gamma(an, bm) \text{ is defined}\}.$$

$\Sigma(r(P, Q)) = P + Q$ stands for

$$\Sigma(r(pn, pm)) = i\_pn\_sig(pn) + i\_pn\_sig(pm)$$

[¬IE] $r(P0, Q0) \cdot R0 = R0$ if $R0 \neq P0$ and $R0 \neq Q0$ stands for
$\{[\neg IE]\, r(pn, pm) \cdot pt = pt \mid 1 \leq n, 1 \leq m, n \neq t \neq m\}$

5 BMACP(A, γ, N), THE AXIOMS

5.1. The first two axioms determine the effect of the communication function on atoms.

[0] $a0 \mid b0 = \gamma(a0, b0)$ if $\gamma(a0, b0)$ is defined
[1] $a0 \mid b0 = \delta$ if $\gamma(a0, b0)$ is undefined

5.2. The axioms of ACP from [BK 84] minus the axiom that says that δ is a left zero for multiplication, which is modified into [8].

[2] $X + Y = Y + X$
[3] $(X + Y) + Z = X + (Y + Z)$
[4] $X + X = X$
[5] $(X + Y) \cdot Z = X \cdot Z + Y \cdot Z$
[6] $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$
[7] $\delta + X = X$
[8] $\delta \cdot X = \partial(X)$
[9] $a \mid b = b \mid a$
[10] $(a \mid b) \mid c = a \mid (b \mid c)$
[11] $\delta \mid a = \delta$
[12] $X \parallel Y = X \between Y + Y \between X + X \mid Y$
[13] $(a \cdot X) \between Y = a \cdot (X \parallel Y)$
[14] $a \between Y = a \cdot Y$
[15] $(X + Y) \between Z = X \between Z + Y \between Z$
[16] $(a \cdot X) \mid b = (a \mid b) \cdot X$
[17] $a \mid (b \cdot X) = (a \mid b) \cdot X$
[18] $(a \cdot X) \mid (b \cdot Y) = (a \mid b) \cdot (X \parallel Y)$
[19] $X \mid (Y + Z) = X \mid Y + X \mid Z$
[20] $(X + Y) \mid Z = X \mid Z + Y \mid Z$
[21] $\partial_H(\delta) = \delta$
[22] $\partial_H(a0) = a0$ if $a0 \notin H$
[23] $\partial_H(a0) = \delta$ if $a0 \in H$
[24] $\partial_H(X + Y) = \partial_H(X) + \partial_H(Y)$
[25] $\partial_H(X \cdot Y) = \partial_H(X) \cdot \partial_H(Y)$

In view of equation 8 we need additional axioms for process removal in connection with ACP. (In

the present context the identity $\partial(X) = \partial_A(X)$ holds. The introduction of process removal, suggested to us by F.Vaandrager, is more systematic however because it allows extensions of the calculus where encapsulation and process removal will diverge.)

[26] $\partial(a) = \delta$

[27] $\partial(X + Y) = \partial(X) + \partial(Y)$

[28] $\partial(X \cdot Y) = \partial(X) + \partial(Y)$

[29] $\partial(X \parallel Y) = \partial(X) + \partial(Y)$

[30] $\partial(X \between Y) = \partial(X) + \partial(Y)$

[31] $\partial(X \mid Y) = \partial(X) + \partial(Y)$

[32] $\partial(\partial_H(X)) = \partial(X)$

[33] $\partial(P) = T(P)$

[34] $\partial(T(V)) = T(V)$

[35] $\partial(\langle P = X \rangle) = \langle P = X \rangle$

[36] $\partial(V \,\square\, X) = V \,\square\, \partial(X)$

5.3. Axioms concerning the boolean algebra SIG.

[37] $\emptyset + U = U$

[38] $U + V = V + U$

[39] $(U + V) + W = U + (V + W)$

[40] $U + U = U$

[41] $\emptyset \cap U = \emptyset$

[42] $U \cap V = V \cap U$

[43] $(U \cap V) \cap W = U \cap (V \cap W)$

[44] $U \cap U = U$

[45] $P0 \cap Q0 = \emptyset$ **if** $P0 \neq Q0$

[46] $(U + V) \cap W = (U \cap W) + (V \cap W)$

5.4. General axioms on renamings and their effects.

[47] $r(P, Q) = r(Q, P)$

[48] $r(P, P) = Id$

[49] $[\neg IE]$ $r(P, Q) \cdot (P) = Q$

[50] $[\neg IE]$ $r(P0, Q0) \cdot (R0) = R0$ **if** $P0 \neq R0$ **and** $Q0 \neq R0$

[51] $\Sigma(r(P, Q)) = P + Q$

[52] $r \cdot i\_pn\_sig(P) = i\_pn\_sig(r \cdot P)$

[53] $r \cdot \emptyset = \emptyset$

[54] $r \cdot (U + V) = (r \cdot U) + (r \cdot V)$

Renaming commutes with embedding in PM:

[55] $r \cdot i\_pn\_pm(P) = i\_pn\_pm(r \cdot P)$

5.5. The axioms of module algebra from [BHK 86] with some simplifications allowed by the fact that the present concepts of both a signature and an atomic module are simpler than the general ones in [BHK 86] and with some additions due to the presence of process algebra operators.

[56] $\Sigma(X) = \Sigma(\partial(X))$

[57] $\Sigma(\langle P = X \rangle) = P + \Sigma(X)$

[58] $\Sigma(P) = P$

[59] $\Sigma(T(U)) = U$

[60] $\Sigma(X + Y) = \Sigma(X) + \Sigma(Y)$

[61] $\Sigma(U \triangleright X) = U \cap \Sigma(X)$

[62] $\Sigma(r \cdot X) = r \cdot \Sigma(X)$

[63] $r \cdot \langle P = X \rangle = \langle r \cdot P = r \cdot X \rangle$

[64] $r \cdot T(U) = T(r \cdot U)$

[65] $r \cdot (X + Y) = (r \cdot X) + (r \cdot Y)$

[66] $r \cdot (U \triangleright X) = (r \cdot U) \triangleright (r \cdot X)$

[67] $r \cdot (r \cdot X) = X$

[68] $\Sigma(r) \cap \Sigma(X) = \emptyset \rightarrow r \cdot X = X$

[2] $X + Y = Y + X$

[3] $(X + Y) + Z = X + (Y + Z)$

[69] $T(U + V) = T(U) + T(V)$

[70] $X + T(\Sigma(X)) = X$

[71] $X + \partial(U \triangleright X) = X$

[72] $\Sigma(X) \triangleright X = X$

[73] $U \triangleright (V \triangleright X)) = (U \cap V) \triangleright X$

[74] $U \triangleright (T(V) + X) = T(U \cap V) + (U \triangleright X)$

[75] $(\Sigma(X) \cap \Sigma(Y)) = U \rightarrow U \triangleright (X + Y) = (U \triangleright X) + (U \triangleright Y)$

5.6. Additional axioms for renamings.

[76] $r \cdot a = a$

[77] $r \cdot (X \cdot Y) = (r \cdot X) \cdot (r \cdot Y)$

[78] $r \cdot (X \parallel Y) = (r \cdot X) \parallel (r \cdot Y)$

[79] $r \cdot (X \lfloor\!\lfloor Y) = (r \cdot X) \lfloor\!\lfloor (r \cdot Y)$

[80] $r \cdot (X \mid Y) = (r \cdot X) \mid (r \cdot Y)$

[81] $r \cdot \partial_H(X) = \partial_H(r \cdot X)$

5.7. Axioms that determine the effect of process algebra constructors on expressions involving declarations:

[82] $\partial_H(\partial(Z)) = \partial(Z)$

[83] $(X + \partial(Z)) \cdot Y = X \cdot Y + \partial(Z)$

[84] $(X + \partial(Z)) \parallel Y = X \parallel Y + \partial(Z)$

[85] $(X + \partial(Z)) \between Y = X \between Y + \partial(Z)$

[86] $(X + \partial(Z)) \mid Y = X \mid Y + \partial(Z)$

[87] $X \cdot (Y + \partial(Z)) = X \cdot Y + \partial(Z)$

[88] $X \parallel (Y + \partial(Z)) = X \parallel Y + \partial(Z)$

[89] $X \between (Y + \partial(Z)) = X \between Y + \partial(Z)$

[90] $X \mid (Y + \partial(Z)) = X \mid Y + \partial(Z)$

5.8. Axioms that allow to remove declarations and exports from declaration bodies:

[91] $\langle P = X + \partial(Z)\rangle = \langle P = X\rangle + \partial(Z)$

[92] $P \cap \Sigma(X) = \emptyset \rightarrow \langle P = U \,\square\, X\rangle = (U + P) \,\square\, \langle P = X\rangle)$

[93] $P \cap U = P \rightarrow \langle P = U \,\square\, X\rangle = U \,\square\, \langle P = X\rangle$

5.9. Axioms that allow to commute export operators and process algebra operators. There are 4 conditional axioms with the same precondition.

[94] $\Sigma(X) \cap \Sigma(Y) = V \rightarrow V \,\square\, (X \cdot Y) = (V \,\square\, X) \cdot (V \,\square\, Y)$

[95] $\Sigma(X) \cap \Sigma(Y) = V \rightarrow V \,\square\, (X \parallel Y) = (V \,\square\, X) \parallel (V \,\square\, Y)$

[96] $\Sigma(X) \cap \Sigma(Y) = V \rightarrow V \,\square\, (X \between Y) = (V \,\square\, X) \between (V \,\square\, Y)$

[97] $\Sigma(X) \cap \Sigma(Y) = V \rightarrow V \,\square\, (X \mid Y) = (V \,\square\, X) \mid (V \,\square\, Y)$

[98] $\partial_H(V \,\square\, X) = V \,\square\, \partial_H(X)$

5.10. The redundant definition removal axiom.

[99] $P \cap (U + \Sigma(X) + \Sigma(Y)) = \emptyset \rightarrow U \,\square\, (X + \langle P = Y\rangle) = U \,\square\, (X + T(P) + \partial(Y))$

5.11. The body replacement axiom.

[100] $P + \langle P = X\rangle = X + \langle P = X\rangle.$

This completes the description of the axioms of BMACP. If a full specification is to be made in, for instance, ASF [BHK 89] the parameter sets A and N with equality function as well as the power set of A with uniform characteristic function ($\in$) must be specified in advance. This will not

generate serious difficulties. It is currently not clear to us how BMACP can be translated into a term rewiting system complete modulo some permutative reductions. That step is needed if it is to be specified in ASF as an executable algebraic specification. Moreover our description of the specification is rather unmodularised itself, and a coding of it in ASF should address that aspect as well.

6. NORMAL FORMS

6.1. Normal form theorem

NORMAL FORM THEOREM. *Let* M *be a closed module expression, i.e. a closed expression of sort* PM *over* $\Sigma_{BMACP}(A, N)$. *Then there exists a closed module expression* M' *which is provably equal to* M *in* BMACP(A, γ, N) *with the following form:*

$$M' = U \,\square\, (M_0 + \langle P_1 = M_1 \rangle + \ldots + \langle P_n = M_n \rangle + T(P_{n+1} + \ldots + P_{n+m}))$$

where the M_i *are process expressions over* $\Sigma_{ACP}(A, N)$ *and the names in* $T(P_{n+1} + \ldots + P_{n+m})$ *occur in* U *but not in the other summands. (Notice that for* $i, j \leq n$ P_i *and* P_j *may coincide.)*

PROOF. We provide a sketch only. Let us call a name occurrence of P hidden if it occurs within the scope of an operator $U \,\square\, (.)$ in such a way that P is not contained in U. Further two occurrences P' and P'' of the name P have equal scope if every subterm M* of M in which P' occurs in a hidden way also contains P''. Starting from M the first step is to rename all hidden occurrences of names in such a way that whenever a name has two occurrences these have equal scope. As an example of this procedure consider:

$$P \,\square\, (d + \langle Q = a \cdot P + b \rangle) + c \cdot Q.$$

In this expression both occurrences of the name Q fail to have equal scope. Then choose an entirely new name, say R. Now $P \cap \Sigma(r(Q, R)) = \emptyset$ hence:

$$P \,\square\, (d + \langle Q = a \cdot P + b \rangle) = r(Q, R) \cdot (P \,\square\, (d + \langle Q = a \cdot P + b \rangle)) =$$

$$(r(Q, R) \cdot P) \,\square\, (r(Q, R) \cdot (d + \langle Q = a \cdot P + b \rangle)) = P \,\square\, (d + \langle R = a \cdot P + b \rangle).$$

It follows that

$$P \,\square\, (d + \langle Q = a \cdot P + b \rangle) + c \cdot Q = P \,\square\, (d + \langle R = a \cdot P + b \rangle) + c \cdot Q$$

which satisfies the requirements.

Once a term has been obtained in which all occurrences of process names have equal scope it follows that all export operators can be moved to an outermost position and then combined into a single application of the export operator. Thereafter the declarations have to be moved towards the roots of expressions. This is done as follows: on basis of the equations of BMACP it is allowed to replace $C[\langle P = K \rangle]$ by $C[\delta] + \langle P = K \rangle$. Of course this step presupposes the absence of export operators in the context C[ ]. After finitely many of such steps a normal form is obtained.

6.2. A meaning for closed module expressions

Let $K = K(\Sigma_{ACP}(A))$ be a process algebra that has atomic actions A and a communication mechanism corresponding to γ. Suppose that K satisfies the axioms of ACP. Let π be a name not in N. The meaning of a closed module expression in normal form M will be given as a pair $SEM(K, M) = (\Sigma(M), F)$ of $\Sigma(M)$ and an element of $P(\Sigma(M) \cup \{\pi\} \to K)$. This is the powerset of the class of valuations from $\Sigma(M) \cup \{\pi\}$ to K. In other words F is a relation with attributes in $\Sigma(M) \cup \{\pi\}$ and with for each attribute the domain equal to $Dom(K)$. Let $M = U \,\square\, (M_0 + \langle P_1 = M_1\rangle + \ldots + \langle P_n = M_n\rangle + T(P_{n+1} + \ldots + P_{n+m}))$. The first component of $SEM(K, M)$ is just the visible signature of M. For the second component we first consider $M^* = M_0 + \langle P_1 = M_1\rangle + \ldots + \langle P_n = M_n\rangle + T(P_{n+1} + \ldots + P_{n+m})$. The meaning of M^* is the pair $(\Sigma(M^*), F^*)$ with F^* the collection of all valuations σ from $\Sigma(M^*) \cup \{\pi\}$ to K such that

$$K, \sigma \models \langle P_1 = M_1\rangle + \ldots + \langle P_n = M_n\rangle \text{ and } (K, \sigma \models M_0) = \sigma(\pi).$$

F is then obtained from F^* by projection to $\Sigma(M) \cup \{\pi\} = U \cap \Sigma(M^*) \cup \{\pi\}$. We say that M has a well-defined semantics (in the terminology of 1.2: $M \in NMF^*$) if in F the attribute π is functionally dependent on the other atributes in $\Sigma(M)$. In this case we say that $SEM(K, M)$ is functional (of course with respect to K).

(This meaning function has its drawbacks. Another possibility is: $SEM^*(K, M) = U\{SEM(K, M') \mid M'$ a normal form congruent with $M\}$.)

6.3. Nice normal forms

It appears that the kind of 'nicety' one may define depends on the process algebra K. We consider the simplest case: $K = A_n$ the finite projection algebra with depth n. At this point we need the projection operators π_n for natural numbers $n > 0$. Defining equations for the projection operators are as follows (the projection axioms PR):

$$\pi_0(X) = \delta, \qquad \pi_n(X + Y) = \pi_n(X) + \pi_n(Y)$$
$$\pi_{n+1}(a) = a, \qquad \pi_{n+1}(a \cdot X) = a \cdot \pi_n(X)$$

We call the normal form $M = U \,\square\, (M_0 + \langle P_1 = M_1\rangle + \ldots + \langle P_n = M_n\rangle + T(P_{n+1} + \ldots + P_{n+m}))$ A_k-*nice* if for every valuation σ in $SEM(A_n, M)$ and for all finite terms $H_1, \ldots H_{n+m}, H$ over $BPA_\delta(A)$ such that for all i: $1 \le i \le n+m$, $A_k, \sigma \models P_i = H_i$, and $A_k, \sigma \models \pi = H$ the following formal proof exists:

$$ACP(A, \gamma) + PR + \{\pi_k(X) = X\} + \{P_i = H_i \mid 1 \le i \le n+m\}$$
$$\vdash M_0 = H.$$

Concerning the model A_∞ the following definition is plausible: a normal form A_∞-nice if it is A_k-nice for every $k > 1$.

Notice that an A_n -nice specification may contain many redundant equations for instance: $\langle P = a \cdot P \rangle = \langle P = a \cdot P \rangle + \langle P = a \cdot a \cdot P \rangle$. That 'niceness' of a normal form is a non-trivial property related to guardedness of specifications can be understood from the following example. Let $A = \{a, b, c\}$. Then $\emptyset \boxdot a \cdot P = (\emptyset \boxdot a \cdot P) + (\emptyset \boxdot a \cdot P) = (\emptyset \boxdot a \cdot P) + (\emptyset \boxdot a \cdot Q) = \emptyset \boxdot (a \cdot P + a \cdot Q)$. Now $SEM(A_2, \emptyset \boxdot (a \cdot P + a \cdot Q)) \neq SEM(A_2, \emptyset \boxdot a \cdot P)$ because $a \cdot b + a \cdot c$ is in the one and not in the other.

6.4. Soundness theorem

SOUNDNESS THEOREM. *Let* $M1$ *and* $M2$ *be two normal forms. Moreover assume that* $BMACP(A, \gamma, N) \vdash M1 = M2$. *Then* $M1$ *is* A_n*-nice if and only* $M2$ *is* A_n*-nice and if both are* A_n*-nice then* $SEM(A_n, M1) = SEM(A_n, M2)$.

PROOF. Omitted.

REMARKS: (i) One really needs a mechanically verified proof of soundness because when put in use in practice it should be reproved for many extended calculi. Notice that from a methodological point of view one has to consider the possibility that the soundness theorem is wrong or a soundness theorem of useful form cannot be found. If so, what must be done? There are several possibilities.

(i) Correct minor mistakes in the axiom system of BMACP or an extension.
(ii) Restrict the notion of a nice normal form.
(iii) Work relative to another model of process algebra.
(iv) Uncouple module algebra combination and process algebra's alternative composition.

It is not clear to us how long these remedies will work. Indeed more general forms of BMACP result if one or more of the following features are taken into account as well: declarations of atomic actions, declarations of local data types, state operators, process creation, interrupts, signals, τ-abstraction, ε, generalised sums, generalised merges, specific scope rules for input data and temporal logic specifications. The real test for our proposal is that it survives these extensions.

(ii) This version of soundness is based on the interpretation of normal forms in the projective limit model. It would be nicer to have an interpretation based on an arbitrary process algebra. We expect, however, that a variety of different interpretations of BMACP can be found.

(iii) A typical example of the problems with the soundness theorem for BMACP is the following: $\emptyset \boxdot (a \cdot P) = \emptyset \boxdot (a \cdot P) + \emptyset \boxdot (a \cdot P) = \emptyset \boxdot (a \cdot P + a \cdot Q)$. The first and third of expressions have different semantics in the sense of 6.2. The difficulty can be remedied in at least

three ways:

(1) restrict attention to cases where all hidden process names are defined by means of guarded recursion equations and use the fact that these have unique solutions to ensure the validity of expressions of the form SEM(X) = SEM(Y),

(2) use a mechanism of origin tracing like the origin consistency rule of ASF [BHK 89] and COLD [FJKR 87] in order to force renamed versions of the same hidden name to have the same meaning after normalisation,

(3) remove axioms that may introduce duplication of process expressions containing export operators (this motivates the modified axiom system BMACP* below).

7. A WEAKER SYSTEM BMACP*

We describe an axiom system BMACP* that is obtained from BMACP by applying three modifications. The relevant properties of this system are mentioned below.

(i) Restrict all axioms that introduce the multiplication of a PM-variable by replacing that variable by a variable over process names.

[4*] $P + P = P$

[5*] $(X + Y) \cdot P = X \cdot P + Y \cdot P$

[12*] $P \parallel Q = P \between Q + Q \between P + P \mid Q$

[15*] $(X + Y) \between P = X \between P + Y \between P$

[19*] $P \mid (Y + Z) = P \mid Y + P \mid Z$

[20*] $(X + Y) \mid P = X \mid P + Y \mid P$

(ii) Remove axiom [100] and add the inverse body replacement axiom:

[100*] $P \cap \Sigma(X) = \emptyset \rightarrow X = \Sigma(X) \,\square\, (P + \langle P = X \rangle)$

(iii) Add the weak body replacement axioms:

[101] $P + \langle P = a \rangle = a + \langle P = a \rangle$

[102] $P + \langle P = Q + R \rangle = Q + R + \langle P = Q + R \rangle$

[103] $P + \langle P = Q \cdot R \rangle = Q \cdot R + \langle P = Q \cdot R \rangle$

[104] $P + \langle P = Q \parallel R \rangle = Q \parallel R + \langle P = Q \parallel R \rangle$

[105] $P + \langle P = Q \between R \rangle = Q \between R + \langle P = Q \between R \rangle$

[106] $P + \langle P = Q \mid R \rangle = Q \mid R + \langle P = Q \mid R \rangle$

[107] $P + \langle P = \partial_H(Q) \rangle = \partial_H(Q) + \langle P = \partial_H(Q) \rangle$

The objective of BMACP* is to find an axiom system which is weaker than BMACP and still allows a normal form theorem of the required kind and allows to write process expressions (occurring in a normal form) into head normal form in all cases in which this is possible in

BMACP. Moreover BMACP* fails to feature axioms that duplicate (or even triplicate) process variables, in particular the axioms 4, 5, 12, 15, 19 and 20. Due to the absence of these axioms the semantic problems mentioned in 6 (iii) will not occur in connection with BMACP*.

8. EXAMPLES AND REMARKS

8.1. A derivation: $\langle P = X\rangle \between (Q + \langle Q = a\rangle) = (\delta + X\rangle) \between (Q + \langle Q = a\rangle) =$
$\delta \between (Q + \langle Q = a\rangle) + \langle P = X\rangle = \delta \cdot (Q + \langle Q = a\rangle) + \langle P = X\rangle =$
$\partial_A(Q + \langle Q = a\rangle) + \langle P = X\rangle = \partial_A(a + \langle Q = a\rangle) + \langle P = X\rangle =$
$\partial_A(a) + \partial_A(\langle Q = a\rangle) + \langle P = X\rangle = \langle Q = a\rangle + \langle P = X\rangle.$

8.2. Consider the following process $X = a \cdot b^{\omega}$. It can be shown that if one intends to specify this process using guarded recursive equations over the signature of BPA at least two equations are needed, for instance $\langle P = a \cdot Q\rangle + \langle Q = b \cdot Q\rangle$. Using the μ construct this can be done with only one equation $\langle P = a \cdot (\mu Q.\ b \cdot Q)\rangle$. So the style of recursive specification by means of guarded recursion equations leads to more equations than seems necessary. In the formalism of BMACP the following specification can be given. $\langle P = a \cdot (Q + \langle Q = b \cdot Q\rangle)\rangle$. It is useful to introduce the notation $P := X$ as an abbreviation for $P + \langle P = X\rangle$. We call this the definition construct. Examples: $\langle P = a \cdot (Q := b \cdot Q)\rangle$, $a \cdot (Q := b \cdot Q)$. The difference between the definition construct and the usual fixed point operator $fix_P(F(P))$ or $\mu P.\ (F(P))$ is the large scope in which the name P is known. In the definition construct this scope is large in the sense that an explicit hiding (non-export) of the name is needed to restrict the scope. The fix (μ) construct allows α-conversion and does not show the name of the bound variable at all. A simulation of the μ-construct in BMACP can be given as follows: $a \cdot (\mu P.\ b \cdot P) = a \cdot \emptyset \triangleleft (P := b \cdot P)$.

8.3 The following derivable identities are useful:

(i) $\delta = T(\emptyset)$

Proof. $\delta = \delta + T(\Sigma(\delta)) = \delta + T(\emptyset) = T(\emptyset)$.

(ii) if $p \cap \Sigma(X) = \emptyset$ then $X = \Sigma(X) \triangleleft (P + \langle P = X\rangle)$

Proof. $X =_{72} \Sigma(X) \triangleleft X =_{74(\text{and (1) above})} \Sigma(X) \triangleleft (X + T(P)) =_{71(\text{with } U = \Sigma(X))} \Sigma(X) \triangleleft (X + T(P) + \partial(X)) =_{99} \Sigma(X) \triangleleft (X + \langle P = X\rangle) =_{100} \Sigma(X) \triangleleft (P + \langle P = X\rangle)$

8.4. An operational semantics for BMACP is given with the following 2 rules on top of the axioms: $a + X \rightarrow^a \surd$, $a \cdot X + Y \rightarrow^a X + \partial(Y)$.
In order to apply these rules declarations must, when possible, be moved inside the part of a process expression that will not be erased.

8.5. Of course the system BMACP as such is of no practical value. The semantic problem involved in scope control of process names will not occur in isolation. A similar method may however be applied in more complicated circumstances. Then it may be a useful simplification to remove the axioms of process algebra that describe dynamic behavior and the axioms of module algebra that are not essential for normalisation altogether. For instance removing axioms 4, 5, 7 to 23 but introducing associativity and commutativity of merge yields a system BMACP' for which a more appealing form of soundness can be proven.

8.6. The signature of BMACP can be used as an abstract syntax for a specification language. The axioms of BMACP allow 'correct' transformations of this abstract syntax. The need for such transformations (but admittedly not for all transformations of BMACP) can be motivated as follows. An appropriate way to design a specification language for ACP is to simultaneously design an abstract syntax, a vertical syntax with key words etc., a graphical/object oriented syntax and a horizontal (mathematical) syntax.

For the abstract syntax one may use an algebraic format with prefix notation. For the horizontal syntax one will replace many prefix operators by infix and mixfix operators, use mathematical symbols and Greek characters, introduce priorities, implicit type conversion and so on. The vertical syntax will allow the explicit use of type and structure information in headers and footers of sections. The graphical syntax will remove the redundant sequential order information that is unavoidable with both horizontal and vertical representations and allow a very flexible way of presenting type and structure information in both a static and a dynamic way. (Notice that our current presentation of BMACP yields a mixture of an abstract and a horizontal syntax. It can be transformed to an abstract syntax by writing all axioms in prefix notation and removing the implicit embedding convention. It can be turned into a horizontal syntax by adding precise information about priorities, bracketing and implicit embeddings.)

Then it must be defined what it means for two representations, say a graphical representation G and a vertical representation V to 'represent the same specification' i.e. whether or not G and H are 'equivalent'. We propose that this question is to be settled via the abstract syntax. Both representations are first transformed to an abstract representation in a uniform way to be defined in advance. This leads to abs(G) and abs(V). Then G and H are considered equivalent if BMACP proves abs(G) and abs(V) equal.

8.7. Acknowledgements. Frits Vaandrager and Rob van Glabbeek have contributed substantially in the final design of the axiom system.

9. REFERENCES

[deBZ 82] J.W. DE BAKKER & J.I. ZUCKER, *Processes and the denotational semantics of concurrency*, Information & Control 54 (1/2), (1982) 153-158

[BHK 86] J.A. BERGSTRA, J. HEERING & P. KLINT, *Module algebra*, Report CS-R8617, Centre for Mathematics and Computer Science, Amsterdam (1986)

[BHK 89] J.A. BERGSTRA, J. HEERING & P. KLINT, (eds.) *Algebraic Specification*, Addison Wesley, ACM Press Frontier Series (1989)

[BK 82] J.A. BERGSTRA & J.W. KLOP, *Fixed point semantics in process algebra*, Mathematical Centre Research Report, IW 206 (1982)

[BK 84] J.A. BERGSTRA & J.W. KLOP, *Process algebra for synchronous communication*, Information & Control 60 (1/3), (1984) 109-137

[B 88] E. BRINKSMA, *On the design of extended LOTOS, A specification language for distributed open systems*, Ph.D.Thesis, University of Twente (1988)

[FJKR 87] L.G.M. FEYS, H.B.M. JONKERS, C.J.P. KOYMANS & G.R. RENARDEL DE LAVALETTE, *Formal definition of the design language* COLD-K, Technical Report, ESPRIT Project 432 METEOR, Philips Research Eindhoven (1987)

[vGV 88] R.J. VAN GLABBEEK & F.W. VAANDRAGER, *Modular specifications in process algebra—with curious queues*, Centre for Mathematics and Computer Science, Report CS-R8821 (1988).

[H 78] C.A.R. HOARE, *Communicating sequential processes*, Comm.ACM 21 (1978) 666-677

[MV 88] S. MAUW & G.J. VELTINK, *A process specification formalism*, University of Amsterdam, Programming Research Group, Report P8814 (1988)

[M 80] R. MILNER, *A calculus of communicating systems*, Springer LNCS, 92 (1980)

[SP 89] SPECS-Consortium / PTT-RNL, *Definition of MR, Version 1*, SPECS document D.WP5.2 (1989)