Algebra and communicating processes

Algebra and Communicating Processes

Jos C.M. Baeten*
Department of Software Technology,
Centre for Mathematics and Computer Science,
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands.

As an example of the use of algebraic methods in computer science, the theory ACP, dealing with concurrent communicating processes, is described.

1. INTRODUCTION.

Process algebra is the study of concurrent communicating processes in an algebraic framework. As the initiator of this field we consider R. MILNER, with his Calculus of Communicating Systems [M80], which formed the basis for most of the axiom systems in the theory ACP of BERGSTRA & KLOP [BK84, BK85]. The endeavor of process algebra is to treat concurrency theory (the theory of concurrent communicating processes) in an *axiomatic* way, just as for instance the study of mathematical objects as groups or fields starts with an axiomatization of the intended objects. The axiomatic method which concerns us, is algebraic in the sense that we consider structures (also called process algebras by some people) which are models of some set of (mostly) equational axioms; these structures are equipped with several operators. Thus, we use the term *algebra* in the sense of model theory.

There is ample motivation for such an axiomatic-algebraic approach to concurrency theory. The main reason is that there is not one definite notion of *process*. There is a staggering amount of properties which one may or may not attribute to processes, there are dozens of views (*semantics*) which one may have on (a particular kind of) processes, and there are infinitely many models of processes. So an attempt to organize this field of process theories leads very naturally and almost unavoidably to an axiomatic methodology – and a curious consequence is that one has to answer the question "What is a process?" with the seemingly circular answer "A process is something that obeys a certain set of axioms ... for processes". The axiomatic method has proven effective in mathematics and mathematical logic – and in our opinion it has its merits in computer science as wel, if only for its organizing and unifying power.

Next to the organizing role of this set-up with axiom systems, their models and the study of their relations, we have the obvious *computational* aspect. Even more than in mathematics and mathematical logic, in computer science it is *algebra* that counts – the well-known etymology of the word should be convincing enough. For instance, in a system verification the use of transition diagrams may be very illuminating, but especially for larger systems it is evidently desirable to have a formalized mathematical language at our disposal in which specifications, computations, proofs can be given in what is in principle a linear notation (evidenced by [B89]). Only then can we hope to succeed in attempts to mechanize our dealings with the objects of interest. In our case the mathematical language is algebraic, with basic constants, operators to construct larger processes, and equations defining the nature of the processes under consideration. (The format of pure equations is not always enough, though. On occasion, conditional equations and some infinitary proof rules are used.) To be specific: we will always insist on the use of congruences, rather than mere equivalences in the construction of process algebras; this in order to preserve the purely algebraic format.

---

* Partial support received by ESPRIT contract 432, A formal integrated approach to industrial software development (METEOR), and RACE contract 1046, Specification and Programming Environment for Communication Software (SPECS).

A further advantage of the use of the axiomatic-algebraic method is that the entire apparatus of mathematical logic and the theory of abstract data types is at our disposal. One can study extensions of axiom systems, homomorphisms of the corresponding process algebras. One can formulate exact statements as to the relative expressibility of some process operators (non-definability results).

Of course, the present axiomatizations for concurrency theory do not cover the entire spectrum of interest. Several aspects of processes are as yet not well treated in the algebraic framework. The most notable examples concern the real-time behaviour of processes, and what is called *true concurrency* (non-interleaving semantics). Algebraic theories for these aspects are under development at the moment (see e.g. VAN GLABBEEK & VAANDRAGER [GV87]).

In our view, process algebra can be seen as a worthy descendant of 'classical' automata theory as it originated three or four decades ago. The crucial difference is that nowadays one is interested not merely in the execution traces (or language) of one automaton, but in the behaviour of systems of communicating automata. As Milner and also HOARE [H85] have discovered, it is then for several purposes no longer sufficient to abstract the behaviour of a process to a language of execution traces. Instead, one has to work with a more discriminating process semantics, in which also the timing of choices of a system component is taken into account. Mathematically, this difference is very sharply expressed in the equation $x\cdot(y + z) = x\cdot y + x\cdot z$, where + denotes choice and · is sequential composition; x,y,z are processes. If one is interested in languages of execution traces (trace semantics), this equations holds; but in process algebra it will in general not hold. Nevertheless, process algebra retains the option of adding the equation and studying its effect. In fact, one goal of process algebra is to form a uniform framework in which several different process semantics can be compared and related. One can call this *comparative concurrency semantics*.

We bring structure in our theory of process algebra by *modularization*, i.e. we start from a minimal theory (containing only the operators $+,\cdot$), and then we add new features one at at time. This allows us to study features in isolation, and to combine the modules of the theory in different ways.

In the following, we give a survey of the theory ACP (Algebra of Communicating Processes) as introduced in [BK84].

2. BASIC PROCESS ALGEBRA.

Process algebra starts with a given set A of **atomic actions** $a,b,c,\ldots$. These actions are taken to be indivisible and to have no duration. When we describe a certain application, we will have to be specify what are the atomic actions involved. Thus, the set A will form a *parameter* of our theory. Each atomic action is a constant in the theory. Actions can be combined into composite processes by the operators + and $\cdot$. + is **alternative composition**, choice or sum, and · is **sequential composition** or product. Thus, $(a+b)\cdot c$ is the process that first chooses between executing a or b, next executes c and then terminates. Since time has a direction, product is not commutative; but sum is, and in fact it is stipulated that the options (summands) possible in some state of the process form a *set*. Formally, we will require that all processes $x,y,\ldots$ satisfy the axioms in table 1.

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)\cdot z = x\cdot z + y\cdot z$	A4
$(x\cdot y)\cdot z = x\cdot(y\cdot z)$	A5

Table 1. BPA.

We often leave out brackets and the product sign, as in regular algebra. Product will bind more strongly than other operators, sum will bind more weakly. Thus, $xy + z$ means $(x\cdot y) + z$. The theory in table 1 is called Basic Process Algebra or BPA.

We do not include an axiom $x(y + z) = xy + xz$ because the moment of choice in both processes is different, and this difference is important in many applications. For instance, a game of Russian roulette could be described by $spin \cdot click + spin \cdot bang$, but not by $spin \cdot (click + bang)$.

3. TERMINATION.

Let us again consider sequential composition of processes. If in process $x \cdot y$ component x has performed all its actions, and can do nothing more, it has terminated successfully and process y starts. But if process x consists of a number of concurrently operating components, that at some point are all waiting for a communication from another component, then x also cannot perform any more actions, but in such a situation we do not want that y start. In the second case, we say x has terminated unsuccessfully, is in a state of **deadlock**, and no action is possible any more. Thus, we want to distinguish between successful and unsuccessful termination. We use the constant δ for unsuccessful termination. The laws for this constant are in table 2.

$x + \delta = x$	A6
$\delta \cdot x = \delta$	A7

Table 2. Deadlock.

Now we can give a more formal argument for rejection of the law $x(y + z) = xy + xz$: a consequence is $ab = a(b + \delta) = ab + a\delta$, and this means that a process with deadlock possibility is equal to one without. In most applications, it is important to model deadlock behaviour, so this is an unwanted identification.

It sometimes has advantages to also include a special constant for successful termination. For this purpose, the **empty process** ε is often used, with laws $\varepsilon x = x = x\varepsilon$.

4. INTERLEAVING.

If we look at the **parallel composition** $x \| y$ of processes x and y from the outside, we will see that the atomic actions of x and y are *interleaved* or merged in time (since we assume they have no duration). Thus, at each point in time, the next action will either come from x or from y. In order to get a finite axiomatisation for the parallel composition or **merge**, we will use an auxiliary operator $\lfloor\!\lfloor$ (left-merge). $x \lfloor\!\lfloor y$ is just like $x \| y$, but with the restriction that the first step comes from x.

$x \| y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$	M1
$a \lfloor\!\lfloor x = ax$	M2
$ax \lfloor\!\lfloor y = a(x \| y)$	M3
$(x + y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$	M4

Table 3. Interleaving.

The theory with constants A, operators $+, \cdot, \|, \lfloor\!\lfloor$ and axioms in tables 1 and 3 is called PA. Axioms M2 and M3 are actually *axiom schemes*: we have such an axiom for each $a \in A \cup \{\delta\}$. With the axioms of PA, we can eliminate the operators $\|, \lfloor\!\lfloor$ from all closed terms. In fact, this elimination takes the form of a *term rewrite system*. Thus, merge becomes a defined operator on closed terms (but not on infinite processes, defined by means of recursive equations).

5. COMMUNICATION.

Parallel composition between processes is not interesting without some form of communication. For this reason, we extend the merge operator of section 4 to include the possibilities for communication. First, we need to say which atomic actions can communicate, which actions are communication partners. For this reason, we assume we have a **communication function** γ given on the set of atomic actions A. This is a partial binary function on A; if $\gamma(a,b) = c$, we say that a and b communicate, and the result of the communication is c; if $\gamma(a,b)$ is undefined, we say that a and b do not communicate. We

do not restrict ourselves beforehand to binary communication only, but will require that γ is commutative and associative, i.e. for all $a,b,c \in A$ we have

$$\gamma(a,b) = \gamma(b,a)$$
$$\gamma(a,\gamma(b,c)) = \gamma(\gamma(a,b),c),$$

and either side of these equations is defined exactly when the other side is. Now, the parameters of our theory are A and γ.

In order to incorporate the possibility for communication in the merge operator, we use an additional auxiliary operator $|$ (communication merge). Now, $x \mid y$ is just like $x \between y$, but with the restriction that the first step is a communication step between x and y. In table 4, $a,b \in A \cup \{\delta\}$, and x,y,z are arbitrary processes.

$a \mid b = \gamma(a,b)$	if $\gamma(a,b)$ is defined	CF1
$a \mid b = \delta$	otherwise	CF2
$x \between y = x \lfloor y + y \lfloor x + x \mid y$		CM1
$a \lfloor x = ax$		CM2
$ax \lfloor y = a(x \between y)$		CM3
$(x + y) \lfloor z = x \lfloor z + y \lfloor z$		CM4
$ax \mid b = (a \mid b)x$		CM5
$a \mid bx = (a \mid b)x$		CM6
$ax \mid by = (a \mid b)(x \between y)$		CM7
$(x+y) \mid z = x \mid z + y \mid z$		CM8
$x \mid (y+z) = x \mid y + x \mid z$		CM9

Table 4. Merge with communication.

6. ENCAPSULATION.

In communicating systems, we often want that communication partners should communicate, and not occur by themselves. In order to block unwanted occurrences of such actions, we need the **encapsulation operators** ∂_H. Here, H is a set of atomic actions ($H \subseteq A$), and ∂_H will block all actions from H, by renaming them into δ.

$\partial_H(a) = \delta$	if $a \in H$	D1
$\partial_H(a) = a$	otherwise	D2
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$		D3
$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$		D4

Table 5. Encapsulation.

The axioms in table 1,2,4 and 5 together constitute the axiom system ACP (Algebra of Communicating Processes) of BERGSTRA & KLOP [BK84]. Typically, a system of communicating processes $x_1,\ldots,x_n$ is represented in ACP by the expression $\partial_H(x_1 \between \ldots \between x_n)$, where H will contain all communication 'halves' occurring in the parallel composition.

This language (with some extra defined operators) has been used extensively in [1] in system specification. In order to do system *verification*, it is necessary to tackle the issue of *abstraction* as in [BK85].

REFERENCES.

[B89] J.C.M. BAETEN (ed.), *Applications of process algebra,* CWI Monograph 8, North-Holland, Amsterdam 1989. To appear.

[BK84] J.A. BERGSTRA & J.W. KLOP, *Process algebra for synchronous communication,* Inf. & Control 60 (1/3), 1984, pp. 109-137.

[BK85] J.A. BERGSTRA & J.W. KLOP, *Algebra of communicating processes with abstraction,* Theor. Comp. Sci. 37 (1), 1985, pp. 77-121.

[GV87] R.J. VAN GLABBEEK & F.W. VAANDRAGER, *Petri net models for algebraic theories of concurrency,* in: Proc. PARLE II (J.W. de Bakker, A.J. Nijman & P.C. Treleaven, eds.), Springer LNCS 259, 1987, pp. 224-242.

[H85] C.A.R. HOARE, *Communicating sequential processes,* Prentice-Hall, 1985.

[M80] R. MILNER, *A calculus of communicating systems,* Springer LNCS 92, 1980.