What is algebraic in process theory?

What is algebraic in process theory?*

Bas Luttik

Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

CWI, P.O. Box 94079, 1090 GB, The Netherlands.

E-mail: s.p.luttik@tue.nl
www: http://www.win.tue.nl/~luttik.

Abstract

This is an extended version of an essay with the same title that I wrote for the workshop *Algebraic Process Calculi: The First Twenty Five Years and Beyond*, held in Bertinoro, Italy in the first week of August 2005.

1 Prologue

In mathematics, sometimes two kinds of *algebra* are distinguished: elementary and abstract. *Elementary algebra* records the properties of the real number system, mostly in the form of equations using symbols to denote constants (particular real numbers) and variables (ranging over all real numbers). Elementary algebra is concrete in the sense that it is about one particular kind of object: the real number. *Abstract algebra* (also known as *modern algebra*) is concerned with the study of the (properties of the) fundamental operations of arithmetic in more generality, e.g., no longer talking about addition of real numbers only, but talking about addition of anything that might be worth adding. The generality is usually achieved by defining the fundamental operations axiomatically.

For an example of an axiomatic definition, consider a theory of two binary operations defined by postulating that the first operation is commutative, associative and idempotent, and that the second operation distributes from the right over the first and is also associative. A ring theorist may tell you that this comes close to a definition of the theory of idempotent semirings, except that a few axioms are surely missing. Most notably, the ring theorist remarks, an axiom expressing that the second operation also distributes from the left over the first ought to be included. A

*Appeared in: Luca Aceto, editor, The Concurrency Column, Bulletin of the EATCS 88, February 2006.

process theorist will recognize that this axiom has been left out on purpose, for what we have here is a definition of the theory of alternative and sequential composition of processes. This particular version of the theory was proposed by Bergstra and Klop in 1982 (see [10, 12]); they presented it as a set of formal equations.

In this note I will look at process theory from the algebraic perspective. I will indicate what are the algebraic achievements and I'll classify them according to whether they are elementary algebraic or abstract algebraic in the above sense. We shall see that most results in algebraic process theory are elementary, but I'll give two examples of a more abstract nature, which, I hope, demonstrate the advantages of the abstract algebraic approach.

2 Algebraic process theory

Algebraic process theory started in the 1970's, with the introduction of CSP by Hoare [17, 39, 40] and of CCS by Milner [45, 46]. What is algebraic about CSP and CCS? It is the fact that the emphasis is on studying the properties of a collection of fundamental operations on processes. CSP and CCS for the bigger part agree on what are those fundamental operations, both including sequencing, nondeterministic choice, and parallel composition. Moreover, these constructs turned out to satisfy very similar properties. The main difference between CSP and CCS lies in to what is viewed as the appropriate mathematical representation of the notion of process: in CSP a process is mathematically represented as a set of failures[1], whereas in CCS it is an element of the set of labelled transition systems modulo observation equivalence.

Defining a language of first-order operations on a domain of processes and proving properties of these operations is algebra in the elementary sense of the word. With their seminal paper [37], Hennessy and Milner made an important step in the direction of a more abstract approach, providing a complete characterisations of observation equivalence in the context of recursion-free CCS. Their characterisation could in principle be taken as an abstract algebraic definition. A genuinely abstract algebraic approach was first explicitly proposed by Bergstra and Klop [10, 12]. One of their methodological concerns when introducing ACP was to present "first a system of axioms for communicating processes [...] and next study its models" (see [12, p. 112]).

The system of axioms of ACP is presented as a set of formal equations. Table 1 lists the axioms of the fragment of ACP pertaining to the binary operations for *alternative composition* (denoted by $+$), *sequential composition* (denoted by $\cdot$), and the constant *deadlock* (denoted by δ); the fragment is called BPA_δ (Basic Process Algebra with deadlock).

A formal framework for specifying and reasoning about processes is obtained almost for free. The idea is to declare a set *Acts* of constants called *actions*, and specify complex processes from them using process theoretic operations and recursion equations. For instance, the three recursion equations in Table 2 specify the behaviour of

[1] A *failure* is a sequence of events in which a process may engage, together with a set of events that it subsequently refuses to engage in.

Table 1: The axioms of BPA_δ.

(A1)	$p+q$	$=$	$q+p$	(A6)	$p+\delta$	$=$	p
(A2)	$p+(q+r)$	$=$	$(p+q)+r$	(A7)	$\delta\cdot p$	$=$	δ
(A3)	$p+p$	$=$	p				
(A4)	$(p+q)\cdot r$	$=$	$p\cdot r+q\cdot r$				
(A5)	$(p\cdot q)\cdot r$	$=$	$p\cdot(q\cdot r)$				

a stack of bits from four actions $push_0$, pop_0, $push_1$ and pop_1 (representing the basic actions of pushing and popping the bits 0 and 1, respectively). Equational logic with the equational axioms defining the operations (extended with a proof rule to reason about recursive definitions, if necessary) constitutes a convenient deductive system for reasoning about the correctness of a specification.

Let me try to avoid a misunderstanding here as to why Bergstra and Klop's theory is algebraic in the sense of abstract algebra. It is *not* (or at least not merely) the fact that it uses equational axioms to define the operations. The equations are just a means to realise the real desideratum of abstract algebra, which is to abstract from the nature of the objects under consideration. In the same way as the mathematical theory of rings is about arithmetic without relying on a mathematical definition of *number*, Bergstra and Klop's proposal deals with process theory without relying on a mathematical definition of *process*.

Consider the axiomatic theory BPA_δ and an arbitrary nonempty set P endowed with two binary operations and a distinguished element. If we agree to denote one of the binary operations by $+$, the other by $\cdot$, and the distinguished element by δ, then it makes sense to ask whether the axioms of BPA_δ hold in P. For instance, the axiom (A1) holds in P if the operation on P denoted by $+$ is commutative, i.e., if for all elements p and q of P the operation $+$ maps the pairs (p,q) and (q,p) to the same element of P. A *model* of BPA_δ is *any* algebraic structure P with two binary operations $+$ and $\cdot$ and a distinguished element δ such that the axioms of BPA_δ hold for all elements p, q and r of P.

Nowadays, the prime examples of models of ACP and other algebraic process theories are obtained by associating an operational semantics with a set of process expressions, or recursive specifications of the kind presented in Table 2, and dividing

Table 2: The behaviour of a stack of bits

$$
\begin{aligned}
S &\stackrel{\text{def}}{=} push_0\cdot T_0\cdot S + push_1\cdot T_1\cdot S\\
T_0 &\stackrel{\text{def}}{=} pop_0 + push_0\cdot T_0\cdot T_0 + push_1\cdot T_1\cdot T_0\\
T_1 &\stackrel{\text{def}}{=} pop_1 + push_0\cdot T_0\cdot T_1 + push_1\cdot T_1\cdot T_1
\end{aligned}
$$

out bisimulation congruence (see, e.g., [8, 22]). But other interesting models of processes have been defined by choosing a different mathematical representation for the notion of process. For instance, Bergstra and Klop defined a *projective limit* model in [12] and a *graph model* in [13].

A great advantage of the abstract algebraic approach is that there is no need to commit to one particular mathematical representation of the notion of process, before we can start reasoning about processes. By Birkhoff's completeness theorem for equational logic [15], every equation that can be derived from the axioms of BPA_δ by equational reasoning will automatically hold in every model of BPA_δ.

A further advantage is that via the axioms process theory makes contact with other areas of mathematics. Note that the set of natural numbers with addition and multiplication as binary operations and the natural number 0 as distinguished element is also a model of BPA_δ. And a set of propositions with disjunction and conjunction as binary operations and the proposition 0 (false) as distinguished element (or actually any Boolean algebra) is also a model of BPA_δ. Perhaps these connections with number theory and logic seem far-fetched at first, but we shall see later that both connections in fact lead to beneficial theoretical insights.

3 Algebraic achievements

In the second half of the 1980's the algebraic approach in process theory received quite some attention. We briefly mention three categories of algebraic results (see Aceto's column [2] for a more elaborate overview with the appropriate references):

Expressiveness: Several results were obtained showing that certain combinations of fundamental process theoretic operations are more expressive than others. For instance, it was shown that the use of sequential composition is crucial in the specification of the behaviour of the stack of bits in Table 2; it cannot be specified by means of a finite recursive specification if prefix multiplication is to be used instead of sequential composition [11].

Axiomatisability: A lot of effort was put into providing satisfactory equational axiom systems for certain combinations of process theoretic operations, and showing that satisfactory equational axiom systems do not exist for other combinations (see the survey [3]). Here *satisfactory* usually meant *finite* and *ground-complete*[2] with respect to some notion of behavioural congruence.

Unique decomposition: For several versions of parallel composition a unique decomposition theorem was established to the effect that every process can be uniquely expressed as a parallel composition of parallel prime processes up to a certain behavioural equivalence. The first such result was obtained by Milner and Moller in [47].

[2] An axiomatisation is *ground-complete* if any two behaviourally equivalent *closed* process expressions are provably equal.

Most of the results mentioned above are algebraic in the same way as elementary algebra is algebraic: they record properties of a collection of operations defined on a predetermined mathematical model of processes (usually, labelled transition systems modulo a behavioural equivalence). The ω-completeness[3] results presented by Moller [48] in his excellent PhD thesis can be considered an exception; they are more abstract algebraic since they are about the quality of the axiom systems themselves and do not rely on a particular predetermined mathematical model of processes.

4 Process theoretic binders

At the end of the 1980's, the algebraic process theories were no longer exclusively used for theoretical studies of the properties of fundamental operations on processes. Attempts were made to employ them as formal frameworks for the compositional specification and verification of realistic systems. However, it was found that, for that purpose, they were lacking expressiveness. It was, for instance, too cumbersome to succinctly specify complex systems, because the data that was passed around in these systems could not be handled formally. To obtain practically applicable process specification languages based on the original algebraic process theories, many sophisticated features such as data, time, mobility, probability and stochastics were introduced [6], and less effort was put into providing a proper algebraic treatment for these features.

Many of the advanced features were added in the form of variable binding operations. Unlike, e.g., the operations of alternative and sequential composition, a binder is not directly suitable for abstract algebraic treatment. The reason is that it relies on the syntactic nature of the object on which it acts, thereby going against the main desideratum of abstract algebra that the intrinsic nature of the objects should not matter. In the remainder of this section I will elaborate on this, taking as an example the variable binding *choice quantifiers* from the process specification language μCRL [28], which combines process theoretic operations from ACP with a facility to formally specify abstract data types, as an example.

Choice quantifiers for process specification

Let's return to the specification of the behaviour of a stack of bits in Table 2. The bits themselves occur in the names of the actions and the recursion variables, but they have no formal status. The implicitness of the data in our specification of the stack of bits is not a very big problem, since the amount of data involved is small. However, as the process specifications and the included data types get more complex, the implicitness is inconvenient, and puts a limit on what can be specified. For instance, a stack of natural numbers could not be specified in the same way because it would require infinite alternative compositions on the right-hand sides of infinitely many equations.

[3] An axiomatisation is *ω-complete* if an equation of open terms is provably equal whenever all of its closed substitution instances are.

Table 3: The behaviour of a stack of integers

$$
\begin{array}{ll}
S & \stackrel{\text{def}}{=} \sum_x push(x) \cdot T(x) \cdot S \\
T(x) & \stackrel{\text{def}}{=} pop(x) + \sum_y push(y) \cdot T(y) \cdot T(x)
\end{array}
$$

Table 4: Proof-theoretic axioms for choice quantification.

(CQ1)	$\sum_x p$	$=$	p	if $x \notin \mathrm{FV}(p)$
(CQ2)	$\sum_x p$	$=$	$\sum_y p[x := y]$	if $y \notin \mathrm{FV}(p)$
(CQ3)	$\sum_x p$	$=$	$\sum_x p + p[x := d]$	
(CQ4)	$\sum_x (p+q)$	$=$	$\sum_x p + \sum_x q$	
(CQ5)	$\sum_x (p \cdot q)$	$=$	$(\sum_x p) \cdot q$	if $x \notin \mathrm{FV}(q)$

The process specification language μCRL offers a facility to specify abstract data types by means of an algebraic specification [9]. Data expressions may occur explicitly in conditionals and as parameters of actions and recursion variables. Furthermore, to specify infinite alternative compositions, μCRL includes choice quantifiers $\sum_x$. The intuition is that if $p(x)$ is a μCRL process expression with a free variable x ranging over the values of some datatype, then $\sum_x p(x)$ denotes an alternative composition with a summand $p(d)$ for every value d of the datatype. The process part of a μCRL specification of a stack of integers could consist of the two equations in Table 3. The data variables x and y now have a formal status: in the full μCRL-specification they would be declared as variables of an abstract datatype of natural numbers, a specification of which would also be included. (I deviate slightly from μCRL's official syntax and terminology; the survey [29] offers preciser details about specification and verification in μCRL.)

Algebraic semantics of choice quantification

From the point of view of process specification, μCRL is a reasonably natural extension of ACP, but unfortunately its semantics and its verification capabilities are much less elegant and straightforward. Table 4 lists a few axiom schemata pertaining to the choice quantifiers. Note how they depend on two inherently syntactic notions:

1. If p is a process expression, then $\mathrm{FV}(p)$ denotes the set of data variables with a free occurrence in p.

2. If p is a process expression, x is a data variable and d is a data expression, then $p[x := d]$ designates the process expression in which all free occurrences of x have been replaced by d. As usual, care should be taken that variables

in d are not captured by choice quantifiers. We circumvent the problem by asserting that the mere writing of the expression $p[x := d]$ implicitly entails that the substitution is correct. So, the axiom schema (CQ3) generates an equation for every combination of a process expression p, a data variable x, and a data expression d such that the substitution of d for x is correct for p.

Especially this latter intricate notion of substitution complicates the reasoning considerably.

The issue of formal reasoning with binders has received a great deal of attention in the past years. Fiore, Plotkin and Turi [20] provided a (categorical) algebraic view on abstract syntax with binders; it was applied in [21] to provide abstract rule formats for the operational semantics value-passing process calculi. Gabbay and Pitts [27] proposed a model of syntax involving binders using FM-sets and, on the basis of that model, developed their *nominal techniques* [50, 26]. Miller and Tiu [44] presented an approach to deal with *generic judgments* in proof theory, and apply it to the operational semantics of the π-calculus. All of these approaches make formal reasoning with binders more tractable by extending the mathematical apparatus to elegantly deal with the problems of binding.

We shall now proceed to outline an algebraic semantics of choice quantification that does *not* require any extension of the mathematical apparatus. We draw inspiration from the techniques (e.g., by Tarski [52]) already developed in the 1950's and 1960's in the context of algebraic logic [32, 33]. The idea is to rid the axioms in Table 4 from all references to the inherently syntactic notions; this will make them suitable as an abstract algebraic definition of choice quantification. Here we shall explain the method under the simplifying assumption that there is only a single data variable x; at the end of the section we briefly indicate how it can be made to work also in a setting with an unbounded supply of data variables.

Note that we could in principle try to abstract from the binding qualities of the choice quantifier simply by treating it as a unary operation. That is, let P be a nonempty set endowed with binary operations $+$ and $\cdot$, a unary operation $\sum_x$, and a distinguished element δ. It does not make sense to ask whether P satisfies the axioms in Table 4, unless the elements of P are process expressions for which appropriate notions of *free variable* and *substitution* are defined. Nevertheless, the axioms in Table 4 do have something to say also about an abstract notion of choice quantification.

Consider the proviso $x \notin \mathrm{FV}(p)$, and note that there are two special situations in which we can be certain that it is true: the first is when $p = \delta$, and the second is when $p = \sum_x q$ for some q. This insight can be used to eliminate the provisos $x \notin \mathrm{FV}(p)$ and $x \notin \mathrm{FV}(q)$ from the axioms:

1. replace the axiom (CQ1) by two special instances that need no provisos, viz. $\sum_x \delta = \delta$ and $\sum_x \sum_x p = \sum_x p$; and

2. replace the axiom (CQ5) by a special instance that needs no proviso, viz. $\sum_x (p \cdot \sum_x q) = (\sum_x p) \cdot (\sum_x q)$ (the other instance $\sum_x (p \cdot \delta) = (\sum_x p) \cdot \delta$ is then derivable using $\sum_x \delta = \delta$).

Table 5: Algebraic axioms for (monadic) choice quantification.

(CQ1)′	$\sum\delta$	$=$	δ
(CQ2)′	$\sum\sum p$	$=$	$\sum p$
(CQ3)′	$p+\sum p$	$=$	$\sum p$
(CQ4)′	$\sum(p+q)$	$=$	$\sum p+\sum q$
(CQ5)′	$\sum(p\cdot\sum q)$	$=$	$(\sum p)\cdot(\sum q)$

Substitution can be eliminated replacing (CQ3) by the weaker $\sum_x p = (\sum_x p) + p$ (the axiom (CQ2) has become superfluous by our assumption that x is the only data variable.)

In Table 5 we have listed the axioms for abstract *monadic* choice quantification (the adjective 'monadic' refers to the fact that quantification is over a single data variable). There is a striking coincidence with the axioms of existential quantification as given by Halmos' theory of monadic Boolean algebras [30]; we get them by simply renaming δ to 0, $+$ to $\vee$, $\cdot$ to $\wedge$, and $\sum$ to $\exists$. So, from an abstract algebraic point of view, choice quantification in μCRL is existential quantification in disguise!

In algebraic logic there are two approaches known to generalise the monadic theory to a polyadic theory dealing with quantification over more than one variable. There is the approach of Halmos' *polyadic algebras* [31], which are Boolean algebras extended with a family of quantifiers indexed by subsets of a set I, and a family of transformation operations indexed by mappings from I into I (which would nowadays be called 'explicit substitutions'). There is also the approach of Tarski's *cylindric algebras* [35, 36], which are Boolean algebras extended with a family of quantifiers (called *cylindrifications*) indexed by elements of a set I and a family of distinguished elements indexed by pairs of elements of I called *diagonal elements*. In both cases, the index set I can be thought of as the set of variables. The transformation operations of polyadic Boolean algebras then intuitively correspond to variable renamings; the diagonal elements of cylindric algebras can be thought of as equalities between variables.

For choice quantification the second approach has been worked out in detail in [41]. There the notion of *basic process module* is proposed, which is an algebraic structure endowed with the operations of BPA_δ, a family of unary choice quantifiers (called *projective summations*), and a family of unary operations indexed by the elements of a cylindric algebra (called *guarded commands*); the family of projective summations and the family of cylindrifications of the cylindric algebra are indexed by the same set I. The theory of basic process modules provides an abstract algebraic account of the theory of *parametrised processes*. By declaring a set of constants called *parametrised actions* and specifying their arities using choice quantification, a formal system is obtained that has the same expressive and deductive power as the equational theory of pCRL, a fragment of μCRL. The theory of basic process modules thereby is the abstract algebraic semantics, in the sense of [16], of this fragment of

Table 6: Structural operational rules for $\|$.

$$\frac{E \xrightarrow{a} E'}{E \,\|\, F \xrightarrow{a} E' \,\|\, F} \qquad \frac{F \xrightarrow{a} F'}{E \,\|\, F \xrightarrow{a} E \,\|\, F'}$$

μCRL.

5 Axiomatising proofs

Perhaps one of the reasons why the abstract algebraic approach has been somewhat neglected since the 1990's, is that its advantages were not fully exploited. There is by now a great diversity of process algebras and process specification languages that apart from all their differences also have much in common (e.g., virtually all of them have a binary operation for parallel composition). Then it is likely that certain standard properties need to be established over and over, by proofs that differ on subtle points but nevertheless share many technicalities and insights.

One of the benefits of a well worked out abstract algebraic theory is that it offers a general framework in which to axiomatise such proofs: the commonalities are proved once and for all from the axioms, and the distinguishing details are deferred to the proofs that the axioms hold in concrete cases. Then, to prove the theorem for yet another concrete case, it would suffice to establish that the axioms hold. Below I'll illustrate the idea presenting an axiomatisation of a standard technique used several times to prove the unique parallel decomposition property in subtly different circumstances.

Unique parallel decomposition

The first unique parallel decomposition result was proved by Milner and Moller (see the article [47]). For a set of process expressions $\mathcal{E}$ built from a constant symbol $\mathbf{0}$, unary action prefixes $a.$ ($a \in Acts$) and binary operations $+$ and $\|$, with an associated operational semantics that implemented pure interleaving for $\|$ (see Table 6), they established that every process expression is bisimilar to a parallel composition of parallel prime process expressions that is unique up to bisimilarity and up to the order of the parallel components. Denoting bisimilarity by $\sim$, a process expression E is defined to be *parallel prime* if $E \not\sim \mathbf{0}$ and $E \sim F \,\|\, G$ implies $F \sim \mathbf{0}$ or $G \sim \mathbf{0}$.

Moller extended the result in his dissertation [48], adding the CCS communication facility to the operation for parallel composition. He also discussed to what extent unique parallel decomposition is maintained if the result of communication is treated as unobservable. Christensen in his dissertation [18] further extended the result by admitting a certain type recursive specifications as process expressions, thereby relaxing the assumption that P consists of finite processes to the assumption that it consists of weakly normed processes (processes that have at least one terminating

trace).

Especially when the operation for parallel composition cannot be eliminated, unique parallel decomposition is a very convenient property to have. So, in the literature on action refinement and true-concurrency semantics we find many unique parallel decomposition results [1, 5, 18, 4, 24, 25]. Unique parallel decomposition is also a crucial tool, e.g., in the proofs that bisimilarity is decidable for normed BPP [19] and normed PA [38].

Axiomatising a proof by Milner

The proofs of the unique parallel decomposition results by Moller and Christensen proceed via adaptations of a proof that was discovered by Milner. The part of Milner's proof that establishes uniqueness is by deriving a contradiction from the assumption that an expression has two distinct decompositions; there are two main cases, and the methods for deriving the contradictions in these cases are essentially different and, moreover, require more case ramifications. The proof is very ingenious, but it is also rather technical. Moreover, the technical details of the proof add little insight, and it would be nice if could avoid having to repeat them in subsequent papers. One way to achieve this, is to 'axiomatise' the proof, by carrying out the following steps:

1. reformulate the concrete property (here: unique parallel decomposition up to bisimilarity for the expressions in $\mathcal{E}$) as abstractly as possible;

2. identify the concrete ingredients that make the concrete proof work;

3. reformulate the concrete ingredients in the abstract setting, and carry out an abstract version of the concrete proof, establishing the abstract version of the property from the abstract version of the ingredients.

Below I'll discuss each of these steps for Milner's proof of the unique parallel decomposition property.

Step 1 Instead of with a concrete set of process expressions, considered modulo bisimulation, we prefer to work with an abstract set P of processes. To be able to formulate the unique parallel decomposition property for P, it is of course necessary that there is a binary operation $\|$ for parallel composition defined on it. An element $p \in P$ is called *indecomposable* if it cannot be written as a nontrivial parallel composition. Since P may contain an identity element with respect to parallel composition, i.e., an element $\mathbf{0}$ such that $p = p \,\|\, \mathbf{0}$ for all $p \in P$; the qualification *nontrivial* is necessary to exclude the case that one of the components is this identity element $\mathbf{0}$. It is convenient to proceed with the assumption that P indeed contains an identity element $\mathbf{0}$ for parallel composition.

Now, P is said to have *unique decomposition* if every process in P can be written as a parallel composition of parallel prime processes in P, up to a permutation of the components. The reason for considering uniqueness up to a permutation in the concrete case is that permuting the components of a parallel composition

preserves bisimilarity. In the abstract case, we therefore need to require that parallel composition is a commutative and associative operation.

Note that for the definitions in the preceding two paragraphs it is not important that the elements of P were called *processes* and that the binary operation was called *parallel composition*. What is important, is that P is a *commutative monoid*, i.e., a nonempty set endowed with an associative and commutative binary operation that has an identity element in the set. This completes the first step: we have reformulated the concrete property of unique parallel decomposition as the abstract property of unique decomposition for arbitrary commutative monoids.

Step 2 The ingredients of Milner's proof can all be expressed in terms of the labelled transition relation defined on process expressions by the operational semantics. Actually, the actions never really play a rôle in the proof, so it is convenient to forget about them, writing $E \to F$ if $E \xrightarrow{a} F$ for some $a \in Acts$. A first ingredient that is used, is the following immediate consequence of the definition of bisimilarity:

1. If $E \sim F$ and $E \to E'$, then there exists F' such that $F \to F'$ and $E' \sim F'$.

Milner's proof proceeds by induction on the *size* $|E|$ of E, defined as the length of the longest transition sequence that it affords. The following properties of size play a rôle:

2. $E \sim F$ implies $|E| = |F|$,
3. $|E \parallel F| = |E| + |F|$,
4. $|E| = 0$ iff $E = \mathbf{0}$, and
5. if $|E| > 0$, then there exists E' such that $E \to E'$ and $|E| > |E'|$.

Remarkably, to arrive at a complete description of the ingredients, just one further property of the transition relation $\to$ is needed:

6. if $E \parallel F \to^* G$, then there exist E' and F' such that $E \to^* E'$, $F \to^* F'$ and $G = E' \parallel F'$.

(Note that this last property is a straightforward consequence of the operational rules for $\parallel$ listed in Table 6.)

The reason why Moller's adaptation works, is that these properties continue to hold when CCS-like communication is added. The reason why Christensen's adaptation works, is that these properties also hold if size is defined as the length of the *shortest* transition sequence.

Step 3 The first two ingredients mentioned in the previous step simply say that the transition relation and the size function behave well with respect to bisimilarity; hence, they induce a binary relation and a size function on the set of congruence classes of process expressions modulo bisimilarity. The remaining four ingredients

can then be reformulated as properties of congruence classes of expressions. Conversely, for an arbitrary commutative monoid P with binary operation $\|$ and identity element $\mathbf{0}$, the ingredients of Milner's technique suggest as sufficient condition for unique decomposition: the existence of a binary relation $\rightarrow \subseteq P \times P$ and a size function $|\_| : P \rightarrow \mathbb{N}$ such that the ingredients 3–6 hold (with E and F ranging over P).

It is convenient to translate the abstract ingredients into more standard terminology. In [42] the notion of *decomposition order* is put forward, which is a reformulation of the ingredients of Milner's proof as a combination of standard properties of a partial order $\leq$ on P (well-foundedness, identity the least element, strict compatibility, precompositionality and Archimedeanity). It is proved, by an abstract version of Milner's proof, that the existence of a decomposition order on a commutative monoid is a necessary and sufficient condition for unique decomposition.

Applications

By the theorem in [42], to prove unique parallel composition with respect to *some* version of parallel composition defined on *some* domain of processes, it suffices to establish that the induced commutative monoid can be endowed with a decomposition order. The technicalities of Milner's proof need not be repeated, for they are already dealt with in the proof of the theorem in [42]. I expect that the theorem has particularly straightforward applications proving unique parallel decomposition results in a true-concurrency setting, simplifying, e.g., the proof of a recent unique decomposition result in [25].

The principal example of a commutative monoid with unique decomposition is of course not a set of processes with parallel composition, but the set of positive natural numbers with multiplication. It has unique decomposition by Euclid's fundamental theorem of arithmetic, stating that every positive natural number can be uniquely expressed as product of prime numbers (see, e.g., [34]). The abstract algebraic framework allows us to compare the process theoretic proof of unique decomposition to the number theoretic proof. It turns out that the process theoretic proof is essentially different from its number theoretic counterpart, and it appears that it is a bit more general. For instance, the axiomatisation arising from it can be used to establish the fundamental theorem of arithmetic (although the proof is longer than usual), but the axiomatisation arising from the number theoretic proof is not directly applicable in process theory (see [42] for more detailed discussion).

6 Concluding remarks

I believe that a thorough abstract algebraic treatment will add a degree of mathematical maturity and elegance to the field of process theory. Therefore I think that we should further develop the abstract algebraic side of process theory, by giving abstract algebraic treatments of the advanced process theoretic concepts developed in the 1990's, by finding satisfactory axiom systems, and by trying to prove our results in as general a setting as possible.

Let me end this column with some suggestions for future research that would support the abstract algebraic approach.

At the basis of an abstract algebraic approach lies a thorough understanding of the equational theories of the extant process algebras. For many of them a finite ground-complete equational axiomatisation has been found, or it has been proved that such an axiomatisation does not exist. For a more complete description of the equational theories, we should also consider equations of open terms, and try to find ω-complete axiomatisations. We refer to the survey [3] for a more elaborate discussion and some open problems.

It would be interesting to see whether an abstract algebraic treatment of the constructs of the π-calculus is possible in the same way as we did for choice quantification (i.e., without extending the mathematical apparatus for dealing with the binders). This would then also yield an abstract algebraic definition of mobility.

Of some importance in process theory is the notion of *conservative extension*. The extension of a process calculus with a new construct is conservative if two process expressions without the new construct are equivalent in the extended calculus iff they were equivalent in the original calculus. There are several results in the context of Structural Operational Semantics [51] providing sufficient conditions for when an extension of a process calculus is conservative (see, e.g., [23, 49]). These results are syntactic and, obviously, with a bias towards the operational model. It could be worthwhile to look at conservativity from an abstract algebraic perspective, addressing questions like "Which models of BPA_δ can be conservatively extended with some form of parallel composition?"

There is a proliferation of timed process calculi; the reason is that there are many design decisions to be made about how to incorporate the notion of time (see, e.g., [7]). An abstract algebraic treatment could provide an elegant unifying framework in which such design decisions can be studied and compared.

Acknowledgements

I would like to thank Luca Aceto for his invitation to write this column and I would like to thank Jos Baeten, Simona Orzan and Nikola Trčka for commenting on a draft.

References

[1] L. Aceto. Relating distributed, temporal and causal observations of simple processes. *Fund. Inform.*, 17(4):369–397, 1993.

[2] L. Aceto. Some of my favourite results in classic process algebra (concurrency column). In *Bulletin of the EATCS*, volume 81, pages 89–108. October 2003.

[3] L. Aceto, W. J. Fokkink, A. Ingolfsdottir, and B. Luttik. Finite equational bases in process algebra: results and open questions. In Middeldorp et al. [43], pages 338–367.

[4] L. Aceto, W. J. Fokkink, A. Ingolfsdottir, and B. Luttik. Split-2 bisimilarity has a finite axiomatization over CCS with hennessy's merge. *LMCS*, 1(1:3):1–12, 2005.

[5] L. Aceto and M. Hennessy. Towards action-refinement in process algebras. *Inform. and Comput.*, 103(2):204–269, 1993.

[6] J. C. M. Baeten. A brief history of process algebra. *Theor. Comput. Sci.*, 335(2–3):131–146, 2005.

[7] J. C. M. Baeten, M. R. Mousavi, and M. A. Reniers. Timing the untimed: terminating successfully while being conservative. In Middeldorp et al. [43], pages 281–309.

[8] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[9] J. A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic specification*. ACM Press Frontier Series. Association for Computing Machinery (ACM), New York, 1989.

[10] J. A. Bergstra and J. W. Klop. Fixed point semantics in process algebra. Technical report IW 206, Mathematical Centre, 1982.

[11] J. A. Bergstra and J. W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proceedings of ICALP'84*, LNCS 172, pages 82–95, 1984.

[12] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Inform. and Control*, 60(1–3):109–137, 1984.

[13] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theor. Comput. Sci.*, 37(1):77–121, 1985.

[14] J. A. Bergstra, A. Ponse, and S. A. Smolka. *Handbook of Process Algebra*. Elsevier Science Inc., 2001.

[15] G. Birkhoff. On the structure of abstract algebras. *Proc. Cambridge Philos. Soc.*, 31:433–454, 1935.

[16] W. J. Blok and D. Pigozzi. Algebraizable logics. *Mem. Amer. Math. Soc.*, 77(396), 1989.

[17] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31:560–599, 1984.

[18] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, University of Edinburgh, 1993.

[19] S. Christensen, Y. Hirshfeld, and F. Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *Proceedings of LICS'93*, pages 386–396. IEEE Computer Society Press, 1993.

[20] M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings LICS'99*, pages 193–202. IEEE Computer Society Press, 1999.

[21] M. P. Fiore and D. Turi. Semantics of name and value passing. In *Proceedings of LICS'01*, pages 93–104. IEEE Computer Society Press, 2001.

[22] W. J. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science. Springer, 2000.

[23] W. J. Fokkink and C. Verhoef. A conservative look at operational semantics with variable binding. *Inform. and Comput.*, 146(1):24–54, 1998.

[24] S. B. Fröschle. Composition and decomposition in true-concurrency. In V. Sassone, editor, *Proceedings of FOSSACS'05*, LNCS 3441. Springer, 2005.

[25] S. B. Fröschle and S. Lasota. Decomposition and complexity of hereditary history preserving bisimulation on BPP. In M. Abadi and L. de Alfaro, editors, *Proceedings of CONCUR'05*, LNCS 3653, pages 263–277. Springer, 2005.

[26] M. J. Gabbay and J. Cheney. A sequent calculus for nominal logic. In *Proceedings of LICS'04*, pages 139–148. IEEE Computer Society Press, 2004.

[27] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13:341–363, 2002.

[28] J. F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S. F. M. van Vlijmen, editors, *Algebra of Communicating Processes*, Workshops in Computing, pages 26–62, Utrecht, The Netherlands, 1994. Springer.

[29] J. F. Groote and M. A. Reniers. Algebraic process verification. In *Handbook of Process Algebra* [14], chapter 17, pages 1151–1208.

[30] P. R. Halmos. Algebraic logic, I. monadic boolean algebras. *Compositio Math.*, 12:217–249, 1955.

[31] P. R. Halmos. Algebraic logic, II. homogeneous, locally finite polyadic boolean algebras of infinite degree. *Fund. Math.*, 43:255–325, 1956.

[32] P. R. Halmos. The basic concepts of algebraic logic. *Amer. Math. Monthly*, 63(6):363–387, 1956.

[33] P. R. Halmos and S. Givant. *Logic as algebra*, volume 21 of *The Dolciani Mathematical Expositions*. Mathematical Association of America, Washington, DC, 1998.

[34] G. H. Hardy and E. M. Wright. *An introduction to the theory of numbers*. Oxford University Press, Oxford, Great Britain, fifth edition, 1979.

[35] L. Henkin, J. D. Monk, and A. Tarski. *Cylindric Algebras – Part I*, volume 64 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, 1971.

[36] L. Henkin, J. D. Monk, and A. Tarski. *Cylindric Algebras – Part II*, volume 115 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, 1985.

[37] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, January 1985.

[38] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *Proceedings of ICALP'99*, LNCS 1655, pages 412–421. Springer, 1999.

[39] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978.

[40] C. A. R. Hoare. *Communicating Sequential Processes*. Series in Computer Science. Prentice-Hall International, London, 1985.

[41] B. Luttik. *Choice Quantification in Process Algebra*. PhD thesis, University of Amsterdam, 2002. Available from `http://www.win.tue.nl/~luttik`.

[42] B. Luttik and V. van Oostrom. Decomposition orders—another generalisation of the fundamental theorem of arithmetic. *Theor. Comput. Sci.*, 335:147–186, 2005.

[43] A. Middeldorp, V. van Oostrom, F. van Raamsdonk, and R. C. de Vrijer, editors. *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, LNCS 3838. Springer, 2005.

[44] D. Miller and A. Tiu. A proof theory for generic judgments. *ACM Trans. Comput. Log.*, 6(4):749–783, 2005.

[45] R. Milner. *A Calculus of Communicating Systems*. LNCS 92. Springer, 1980.

[46] R. Milner. *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs, 1989.

[47] R. Milner and F. Moller. Unique decomposition of processes. *Theor. Comput. Sci.*, 107:357–363, January 1993.

[48] F. Moller. *Axioms for Concurrency*. PhD thesis, University of Edinburgh, 1989.

[49] M. R. Mousavi and M. A. Reniers. Orthogonal extensions in structural operational semantics. In *Proceedings of ICALP'05*, LNCS 3580, pages 1214–1225. Springer, 2005.

[50] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Inf. Comput.*, 186(2):165–193, 2003.

[51] G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60–61:17–139, 2004.

[52] A. Tarski. A simplified formalization of predicate logic with identity. *Arch. Math. Logik Grundlagenforsch.*, 7(3–4):61–79, 1965.