The variety of process algebra (DRAFT)

ESPRIT Basic Research Action No 3006

CONCUR

Theories of Concurrency:
Unification and Extension

TASK 1.1
Comparing Process Algebras

Second Year Deliverables

Covering the period
1 September 1990
31 August 1991

Date: September, 1991

Editors: Robin Milner Edinburgh
Faron Moller Edinburgh

TASK 1.1: Comparing Process Algebras

In the review after year one, it was decided together with the reviewers to extend task 1.1 into the second year of CONCUR, with the aim of producing a deliverable at the 24 month milestone comparing the process algebras represented in the project. As a direct result, the paper [1] was produced, a truly collaborative effort of the CONCUR partners. We see that the differences are not so easily resolved, and are linked to the choice of definition method — operational, algebraic, model-based. A theory destined for widespread use must have all three kinds of semantics, and be embedded in a hierarchy of design and specification languages. Also, it must be embedded in a hierarchy of complexity, with inclusion or omission of timing, probability, priority, etc. The partners of CONCUR have different and complementary expertise in models, algebra, operational semantics, modal logic, model checking, assertional specification techniques, algebraic reasoning, and also have different and complementary expertise in application, and this gives genuine synergy to our collaborative project.

In addition, this deliverable contains three other papers. The papers [2] and [3] from SICS give a fundamental study of parallelism, by studying networks equipped with just two operators, disjoint parallelism and linking. The paper [4] from CWI studies Hoare logic in a process algebra framework, thereby providing links between two important bodies of theory.

We conclude that the fundamental work in the short BRA CONCUR provides a sound basis for further unification in the second round of BRA.

Jos Baeten

Contents

ESPRIT Basic Research Action No 3006

CONCUR

Theories of Concurrency: Unification and Extension

Deliverable:	1.1 — Comparing Process Algebras
Paper:	1.1.1
Author(s):	J.C. Baeten, J.A. Bergstra, C.A.R. Hoare, R. Milner, J. Parrow & R. de Simone
Organization:	CWI/UAm/UOx/UEd/SICS/INRIA
Title:	*The Variety of Process Algebra (DRAFT)*
Status:	Report
Date:	1991

The Variety of Process Algebra

(DRAFT)

J.C. Baeten (CWI, Amsterdam)
J.A. Bergstra (University of Amsterdam)
C.A.R. Hoare (University of Oxford)
R. Milner (University of Edinburgh)
J. Parrow (Swedish Institute of Computer Science)
R. de Simone (INRIA, Sophia Antipolis)

August 29, 1991

Abstract

This paper addresses the question whether it is possible, or desirable, to integrate the most widely-studied process algebras. It is natural to pose this question for two reasons: first, the algebras are remarkably alike at first glance; second, an integrated single algebra would appear to benefit applications, in particular via integrated software tools for design and analysis.

The main conclusion is that the variety is both necessary and advantageous. It is *necessary* because several discrepancies, though apparently minor, turn out to be due to radical differences among the stances of the algebras, in turn due to differences of purpose. To reconcile these differences would destroy the special advantages of each algebra; somewhat similar to 'integrating' groups, rings and fields, or the reals and the complex numbers. The variety is *advantageous* because a large number of concurrency phenomena (priority, time, mobility, ...) which were not originally treated in the 'standard' algebras are being brought within the algebraic purview, and it is most unlikely that a single stance will be appropriate for all of them.

Nevertheless, algebraic babel must be avoided. The final message of the paper is therefore that, as more algebras are studied, so must they be interconnected all the more by mathematical means such as homomorphism, embedding, derived algebra etc. This will benefit tool-building as well as provide a coherent theory.

1. Introduction

This paper addresses two questions: First, is it possible to integrate the most widely-studied process algebras, or is their variety *necessary*? Second, if their in-

tegration is possible (perhaps with some adjustment of stance) then is it desirable, or is the variety *advantageous*?

There is a well-established division of labour in science. Traditionally, it is that task of a theoretical scientist to develop a wide range of plausible but competing theories; experimental scientists will then refute or confirm the theories by observation and experiment. In computing science, we can divide the work in an analogous way. Theoretical computing scientists develop a wide range of plausible theories covering a variety of computational paradigms. The mathematical consequences of each theory are explored, and also relationships with other competing or complementary theories. The experimental computing scientist can then select some combination of related theories as the basis for the design of a software system or language, or an architecture for a computing device. The efficiency and effectiveness of this design is then tested in representative case studies. Reliable use of the design will be further assisted by mathematical theorems, methods and heuristics derived from the original theoretical model. And finally, the good (or bad) performance of the design in the field tends to confirm (or to refute) the original theory.

We wish to consider the field of process algebras in this general light. We discuss particular details in this paper, just for the purpose of assessing where we stand in this process of scientific development. As the reader will discover, we believe that this particular field is much larger than first envisaged, and that variety of stance must be maintained. We shall assume that the reader knows something about process algebra already, but not necessarily in great detail.

The algebras which we consider in this paper are ACP [4], CCS [16] and CSP [13]. They are not the only ones, but we confine our attention to them because they are in wide use and because our arguments can be well expressed and illustrated with them. Thus we have only considered MEIJE [1] in the next section, with respect to its status as a 'universal' process algebra. We have omitted Hennessy's EPL, used as a specimen process algebra in his book [9], since it is largely based upon CCS. We have also omitted LOTOS [7], widely used for protocol specification, and CIRCAL [15] used for hardware specification, even though they would indeed provide further interesting contrasts.

The question of integrating process algebras arises naturally in the context of CONCUR, an ESPRIT Basic Research Action whose subtitle is 'Theories of Concurrency: Unification and Extension'; CONCUR is primarily concerned with process algebra, both in theory and in practice. The question has strong practical implications. Not only are process algebras used industrially, but they are also the basis for various software tools for analysing concurrent systems; therefore any possible homogenization brings the strong benefit of common practice.

Another reason for asking our two questions is that the three algebras consider do indeed share a remarkable amount of common ground. This is not all due to their origin within a single European culture. It appears to be due to a natural consensus upon the most basic constructions needed in building a concurrent sys-

tem. All the algebras take synchronized communication as their primitive (and only) means for process interaction; all adopt the basis idea of a guarded command (which can be traced back to Edsger Dijkstra) as the means by which the environment of a process – i.e. other processes – exercises control over the action of the process; all adopt similar forms of sequencing control, and similar forms of parallel composition in which processes are juxtaposed but provided with the opportunity to interact; all adopt similar devices for *abstraction*, i.e. for hiding from view the internal activity of a system so that it can be seen as a black box by its environment; all of them deal with the passage of values in communication in a similar way. So what remains for disagreement?

It has been a surprise, not the least to those who designed these algebras, that despite so much agreement the remaining differences are indeed significant. That differences *exist* is not surprising, since the algebras were not built with exactly the same concerns; in fact, one of the main points made in this paper is that the *stance* of each algebra is quite distinct from that of the others. But it was a reasonable hope that the differences would be *inessential*, i.e. that these stances would emerge as different views of essentially the same mathematical object, which would then emerge as the (perfect?) process algebra of which ACP, CCS and CSP are merely dialects. This has not occurred. We have found that the differences of stance cause non-trivial technical differences; we show this to be so in the next few sections. These discrepancies can easily be glossed over in superficial presentations of process algebra; but the reader should not infer that the discrepancies themselves are superficial! In fact the conclusion to be drawn from the next few sections is that the discrepancies could only be avoided by extending each algebra to the general framework in which they can be embedded, thus losing the distinct advantages of each one.

At the risk of repetition, we wish to emphasize the following point clearly: the adoption of different stances has been in no sense a mistake. The mistake – if there is one – has been the belief or expectation that a single process algebra would be best for all purposes.

In our final section we take account of the large number of new phenomena (timing, priorities, mobility, ...) which are now being considered in the algebraic framework. Faced with this rich field, it is quite unrealistic to expect a single monolithic algebra to suffice. We shall further argue that the variety of stance already adopted in defining process algebras, far from being an inconvenience or failure, will in fact become increasingly *advantageous* when we extend the family of algebras to embrace these new phenomena. Thus the future of process algebra, since it cannot be monolithic, must lie in a family of algebras with strong mathematical interconnection.

2. Universal process algebra?

Before looking at the important differences among process algebras, we wish to examine to what extent they can all be regarded as derivable from a single algebra

which is complete or universal in a well-defined sense.

We recall a theorem by de Simone [19], which asserts that all process operators of a certain kind can be expressed in an exact sense (that is, up to strong bisimilarity – which is the strongest congruence we would expect to demand) in terms of either MEIJE or SCCS [16]. This result is important, since it provides us with a firm background.

First, we need to ask whether the operators of our three algebras have the property which allows de Simone's result to be applied; if so then the algebras are indeed derived algebras of MEIJE and SCCS. The property is that the operator should be definable, in terms of labelled transition systems, by transition rules in a constrained (but quite broad) class. The question is quite easy to answer for CCS, and positively, because CCS operators are already defined by transition rules. The answer is not so immediate for ACP and CSP, because operator meaning is given in a different way. But in fact transition rules can be – and have been – given for the standard ACP and CSP operators which are in a proper sense consistent with their present definitions, and which do indeed fall within de Simone's class. So we shall assume for this discussion that all our algebras are indeed derivable from MEIJE/SCCS.

What do we deduce from this positive answer? We certainly cannot deduce that all users of process algebras should switch to MEIJE or SCCS, whatever their good qualities. Universality does not, by itself, make a good design medium. Nor does it imply a *tractable* theory; often a derived or constrained framework has a more tractable theory – consider the theory of finite state automata versus Turing machines. Nor does the universal theory 'include' the special theory in any useful sense; groups are a special case of monoids, but their theory is far richer and different in character. All we can deduce is that the derived algebras will inherit certain robust qualities from the 'universal' algebra. (Research remains to be done on isolating such qualities.)

It is, however, important to compare and contrast the various process algebras which appear to have tractable theories and practical potential; it is also important to find some structure among the various algebras.

In the next few sections we describe how the different stances of ACP, CCS and CSP, we explain why they were adopted, and we show how they have led to different designs and qualities which it would be hard to unite in a framework which is as tractable as any of the three alone. In the concluding section we look at the future of process algebra, taking account of the increasing variety of phenomena which it seeks to model.

3. Stance

Each process algebra was designed from a particular stance; in each case, the stance was determined by particular practical motives and conceptual concerns.

The stance of ACP is perhaps the closest to algebraic tradition. Its purpose was, and remains, to investigate concurrency by modelling each phenomenon (e.g.

parallel activity, causality, deadlock, interaction, ...) as a few algebraic operators; as usual in algebraic modelling, the nature of the phenomenon is captured by a set of equational axioms over these operators. These equations encapsulate the meaning (operational, or behavioural) of the operators. Take for example the interaction between the *choice* operator '+', and the operator ';' meaning *sequential composition*[1]. In ACP the equation

$$(X + Y); Z = X; Z + Y; Z$$

is an axiom, which reflects the intuition that in any choice expression $P + Q$ the choice occurs at the first action of P or Q, and that the first action of $P; Q$ is that of P. By contrast, and by the same intuition, ACP does *not* postulate

$$X; (Y + Z) = X; Y + X; Z$$

Much research into the theory of ACP has been the investigation of different models for the axioms, including the detection of inconsistencies. As far as application is concerned, ACP has focussed upon topics where equational reasoning is appropriate. Two ingredients are distinctive in its stance: first, it is a family of algebras, varying according to the operators and axioms included; second, its aim is to explore the power of equational proof in applications but not to claim its universality.

CCS starts, like ACP, with a collection of operators; its stance, however, has been primarily operational rather than algebraic. Its purpose was to begin with the smallest set of operators capable of modelling a reasonable range of concurrency phenomena, and to exhibit the operational meaning of the operators as directly as possible, by defining a labelled transition system rather than by giving algebraic axioms. Once this is done, congruence relations are defined over process expressions, representing various notions of 'having the same behaviour'. For each such notion, algebraic laws are derived. In particular, the equation $(X + Y); Z = X; Z + Y; Z$ will indeed hold[2], whereas $X; (Y + Z) = X; Y + X; Z$ will not hold in general. Thus CCS arrives more or less at the place where ACP begins; but as we shall see later, the difference of stance has influenced the choice of operators. A distinctive feature of reasoning in CCS is to mix algebraic reasoning with other techniques (such as bisimulation) associated with the underlying transition system.

In CSP, a third stance is adopted. Though algebraic laws may be the first material presented (e.g. in Hoare's book [13]), CSP was designed with a distinguished model in mind: the *failures* model. The purpose of CSP has been to maximize

[1]We wish not to get embroiled in mere notational variety in this paper. It turns out to be most convenient to adopt the notation of ACP for shared operators, unless otherwise stated, except that we use ';' for sequential composition (ACP just uses juxtaposition).

[2]Sequential composition is not a primitive operator of CCS, but can be defined. One fixes a certain atomic action, say '*done*', to represent successful termination; then sequential composition is a defined using a communication upon this action.

the equivalence relation over processes, subject to the constraint that processes with distinct deadlock or divergence properties must be distinguished; the failures model exactly achieves this goal. The failures model is a refinement of the *traces* model; in it, each process expression denotes a set of failures, and each failure (s, R) consists of a trace s (a sequence of observable actions) together with a set R of observable actions all of which can be *refused* by the process after performing s. Then each operator is defined by its effect upon failure sets; it is by reference to this definition that – again – $(X + Y); Z = X; Z + Y; Z$ will hold except in certain extreme cases[3] but $X; (Y + Z) = X; Y + X; Z$ will not. Again, CSP arrives where ACP begins; but again the stance of CSP has affected its choice of operators as we shall see below. In applications of CSP, proof can be done either by equational reasoning, or directly in terms of failure sets. There is a close analogy with boolean algebra, where one can either used the equational axioms or one can argue directly in terms of set membership.

4. Atomic action

All three algebras build processes on the basis of a set A of *atomic actions* (whose members we shall denote by $a, b, \ldots$). The algebras agree in many respects upon the role of these actions: that they are indeed indivisible; that the overall behaviour of a system, and the way its components interact, is entirely described in terms of these actions; and that the interaction among processes entails synchronisation of actions.

At a deeper level, surprisingly strong differences emerge among the algebras in their treatment of action. This difference is already shown in the way synchronization is handled; further difference emerges when we discuss parallel composition in a later section.

In ACP, an action is itself an atomic process. Recalling that the stance of ACP is rather purely algebraic, it is natural to describe the synchronization regime itself described, like everything else, by equational axioms. First a partial function γ, the *communication function*, is fixed; then for any two actions a and b such that $\gamma(a, b) = c$ defined there is a process axiom

$$a \mid b = c$$

where '|' is the *communication merge* operator on processes. Thus ACP (unlike the other algebras) is parametric upon a synchronization regime, via these axioms. Also, the function γ is required to be commutative and associative, so is extended uniquely to finite multisets of actions by setting $\gamma(a, b, c) = \gamma(a, \gamma(b, c))$ etc. Thus arbitrarily many processes can be synchronized upon a single action; for whenever $\gamma(a, b, \ldots)$ is defined, then this action can result from several processes performing $a, b, \ldots$ simultaneously.

[3] In CSP the choice operator is written '$\Box$', and differs slightly from '+'. This point will be taken up in a later section.

In CCS a motivating idea was that for an external observer to *observe* a process is the same as for two processes to *interact* (i.e. to synchronize in an action). Then, since it is natural to consider an observation as involving exactly two participants (observer and observed), it was natural to restrict *all* synchronizations to involve two participants. Thus in CCS there is a distinguished action τ, and otherwise each atomic action a has a complementary atomic action $\overline{a}$; then CCS can be treated (in this respect) as a special case of ACP in which $\gamma(a,\overline{a}) = \tau$, and $\gamma(a,b)$ is defined iff $b = \overline{a}$.

The difference in CSP is more striking. Here it is inappropriate to say that two processes perform different parts (a and b) of a synchronized action $\gamma(a,b)$; instead, it is more correct to say that processes only synchronize when they can 'engage in' the *same* action. So CSP appears to correspond to the case $\gamma(a,a) = a$, with $\gamma(a,b)$ defined iff $a = b$.

These differences in treatment of action are not profound, yet they lend differences of character to reasoning within the algebras. It is certainly not clear that any one of them is consistently preferable. Moreover, to 'unify' them within one algebra would apparently entail the full generality of actions in MEIJE or SCCS, which constitute an Abelian group under $(\gamma, {}^{-})$. One might argue that, at least for a software tool, this generality should be implemented and then the specialised forms derived; unfortunately, it is notoriously hard in interactive reasoning systems to hide the full generality of the underlying mechanism!

5. Choice

All three algebras agree upon the need for *controllable choice* among processes; that is, the need to be able to combine two processes P and Q in such a way that the environment can determine which of the two is activated, discarding the other. The agreement is total, in the case that P and Q are alternatives guarded by an atomic action; the following all mean the same:

$$\begin{aligned} a;X + b;Y \quad & \text{(ACP)} \\ a.X + b.Y \quad & \text{(CCS)} \\ a{\rightarrow}X \mid b{\rightarrow}Y \quad & \text{(CSP)} \end{aligned}$$

Equally, all the algebras agree that – for completeness – process expressions other than guarded alternatives should be allowed to be combined by controllable choice. Here ACP and CCS coincide, and indeed '+' can combine arbitrary process expressions.

In CSP there is a slight difference, which nonetheless appears to be essential given the particular stance of CSP. The basic choice form $(a{\rightarrow}X \mid b{\rightarrow}Y \mid \ldots)$ is not general, so CSP has an extra *general choice* operator ' $[\!]$ ' which is consistent with the basic form (i.e. $a{\rightarrow}X \mid b{\rightarrow}Y = a{\rightarrow}X \,[\!]\, b{\rightarrow}Y$) but differs from '+' in certain extreme cases. This is evidenced by the fact, mentioned in an earlier section, that the equation

$$(X + Y);Z = X;Z + Y;Z$$

always holds in ACP and (after deriving ';') in CCS, but the equation

$$(X \parallel Y); Z = X; Z \parallel Y; Z$$

can fail in CSP when either X or Y can terminate without action.[4]

Why should we obtrude such a minor discrepancy into a general comparison of approaches? Surely it can be eliminated by adjusting the definition in one or other of the algebras? Well, it appears impossible to do so in a theoretically sound way. In fact the discrepancy manifests a significant difference between stances, which is why it is justified to focus upon such a detail. For the precise meaning of ' $\parallel$ ' in CSP is determined by its definition in terms of failures, and the essence of failures is that they are defined with reference only to *observable* actions. This is the consistent CSP stance. By contrast, the equally consistent stances of ACP and CCS demand homogeneous treatment of observable *and unobservable* actions.

Let us look at some other consequences of this difference of stance. To confine attention only to observable actions, as in CSP, yields both benefits and limitations. As a benefit, the algebraic normal form for CSP processes is pleasantly simple due to the absence of τ-actions. As a limitation, reasoning about fairness of computations cannot be done in terms of failures semantics, since such reasoning is dependent upon unobservable actions.

Thus the holding or non-holding (in extreme cases!) of a distributive law, though a minor matter, is a symptom of a striking incomparability of two stances; one cannot get the benefits of both at once.

6. Sequential composition

Sequential composition, and the allied notion of termination, is another issue which separates the stances of the algebras. It is a subtle issue, and we shall not deal with it fully; we shall do enough to separate one algebra – CCS this time – from the other two, but not enough to separate CSP from ACP, although they do indeed differ.

One may view termination in two ways. First, one may regard successful termination as a special property of (the state of) a process; it is observable, but not observed in the way an action is observed. This approach is taken by both ACP and CSP. (The process which has this property and no capability of action is called `skip` in CSP and ε in ACP.[5]) The 'successful termination' property is then used as the basis for axiomatizing ';' in ACP, or for defining it (in terms of failures) in CSP.

The stance of CCS is to treat all observation as action; CCS only contains one basic form of process termination, unlike ACP and CSP which contain two (successful and unsuccessful). This means that 'successful' termination has to be represented

[4]Formally, this means that X or Y possesses the trace $\surd$, successful termination.

[5]In fact ε is not present in standard ACP, but can be added without inconsistency. In standard ACP the axioms for ':' do not require ε.

in CCS by the performance of a particular action chosen by convention. Then a form of sequential composition is definable in terms of parallel composition in CCS; this operator satisfies the obvious laws of sequential composition, and indeed behaves exactly as sequential composition should behave on processes which respect the termination convention (i.e. don't perform further actions after 'termination').

This difference of approach provides a good test-case for the aim of 'unifying' process algebras. The advantages of the two approaches are not strictly comparable; in the case of ACP and CSP one has a smoother (more direct) mathematical treatment of sequencing, while in the case of CCS one has fewer kinds of observable property and fewer operators. Thus the choice between the two approaches cannot be made on firm theoretical grounds. It is highly likely that their relative merits in different applications will differ; there is no best choice.

7. Parallel composition

Parallel composition is the key operator in all the algebras. We shall write it '||' as in ACP and CSP (in CCS it is written '|'). The essence of $P \parallel Q$ is that P and Q run concurrently, acting separately when possible and interacting when possible. For all three algebras, in the case of interactions, 'when possible' means that if P can do a and Q can do b, and $\gamma(a,b)$ is defined, then $P \parallel Q$ can do $\gamma(a,b)$.

Now ACP and CCS also agree upon what 'when possible' means for separate actions; for them, any action possible for P is also possible as a separate action in the context $P \parallel Q$. So for ACP and CCS the meaning of '||' can be expressed by the following labelled transition rules:

$$\frac{P \xrightarrow{a} P'}{P \parallel Q \xrightarrow{a} P' \parallel Q} \qquad \frac{Q \xrightarrow{b} Q'}{P \parallel Q \xrightarrow{b} P \parallel Q'}$$

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{b} Q'}{P \parallel Q \xrightarrow{\gamma(a,b)} P' \parallel Q'}$$

The CCS case is got by writing $\overline{a}$ for b and τ for $\gamma(a,b)$ in the third rule. Note that these rules do not *force* P and Q to synchronize upon a and b; but the synchronization can indeed be forced by applying an encapsulation or restriction operator.

The parallel composition of CSP is different in conception, and can be understood as follows. First, to every process expression P is assigned an *alphabet* $\alpha P \subseteq A$, representing all the actions which it is allowed to perform. (Strictly speaking, one has to provide an alphabet for each *occurrence* of an expression P, since it can be used with different alphabets at different occurrences. The same effect can be gained by providing two alphabets as extra parameters for the parallel composition operator, in the rules shown below.) Then the idea is that, if we can only detect the actions in αP, then the only difference we would see between

P alone and $P \parallel Q$ is that the latter is less active – since P may separately perform any action not in αQ, but requires the engagement of Q in any action in the common alphabet $\alpha P \cap \alpha Q$. Seen through the window αP, the actions of P are only constrained (not altered) by Q, and similarly with P and Q exchanged. As inference rules, this is expressed

$$\frac{P \xrightarrow{a} P'}{P \parallel Q \xrightarrow{a} P' \parallel Q} (a \in \alpha P - \alpha Q) \qquad \frac{Q \xrightarrow{b} Q'}{P \parallel Q \xrightarrow{b} P \parallel Q'} (b \in \alpha Q - \alpha P)$$

$$\frac{P \xrightarrow{c} P' \quad Q \xrightarrow{c} Q'}{P \parallel Q \xrightarrow{c} P' \parallel Q'} (c \in \alpha P \cap \alpha Q)$$

In looking at these three kinds of parallel composition, we find them incomparable, in the strict sense of the word; that is, each appears to have a different advantage, and if we try to weigh one advantage against the other we find that neither outweighs the other absolutely; it all depends on your concerns. The ACP form has greatest expressive power[6], but this is achieved by allowing γ to vary from one application to another. The simplicity of CCS here is gained at the price of only admitting two-way synchronization. The CSP form uses a very simple synchronization structure, and also has the advantage that it preserves determinacy, while the ACP and CCS forms definitely do not; so CSP can claim to have separated non-determinism from concurrency. On the other hand, the CSP parallel operator requires explicit provision of alphabets.

8. Conclusion: the future family of process algebras

The scope of the 'standard' process algebras we have been considering is limited to asynchronous concurrency (with synchronized communication) among nondeterministic untimed processes, whose neighbourhood structure cannot vary with time. Even if this were the whole story, the problem of unifying the algebras into a 'best' algebra from all points of view is insoluble, as we have shown. This is not to say that a more coherent relationship among them cannot be found. Already we know (or strongly believe) that all of them are derived algebras of MEIJE/SCCS; in each of the algebras there has also been considerable study of sub-algebras, derived algebras (the simplest being those which omit some operators), homomorphisms (often expressed in the form of congruence relations), and so on. Nonetheless, further structuring is possible and should be pursued.

But it is *not* the whole story! New concepts, or at least concepts not previously treated algebraically, are continually being brought within the domain of process algebra; and after a decade of 'classical' process algebra of the above kind, this expansion is actually intensifying. Some examples are as follows:

[6] To be precise: the CCS and CSP parallel compositions can be defined in ACP, provided the latter is equipped with a renaming operator. While it does not appear possible to translate the ACP parallel operator (for arbitrary fixed γ) directly into CSP, it is generally easy to reformulate each particular ACP usage in CSP terms.

- Priorities among processes, or among actions;
- Real time, expressed as duration of actions or as interval between actions, exact or constrained by bounds;
- Extension to embrace reasoning about value domains;
- Probabilistic rather than non-deterministic choice;
- Higher-order processes, which can transmit processes as values during communication;
- Mobility among processes, expressed by the transmission of channels (or action-names) in a communication;
- Asynchronous communication.

Many of these new features are indeed studied within the CONCUR project. Examples of recent work are: real time [5, 2, 6, 12, 18]; value domains [5, 11]; mobility [10, 17]; asynchronous communication [3, 14].

In the face of the huge challenge to treat all of these developments in a disciplined way, the highest priority must not be to integrate the (small!) world of classical process algebra, but rather to explore this larger domain both for its own sake and in order to find clues and guidance for how to integrate the variety of process algebras in a coherent family. To give just one example of what should be done on a larger scale: the algebras of timed and untimed (or classical) CSP have been coherently related, in the sense that their mathematical foundation allows results from the untimed algebra to be transferred to the timed algebra.

As far as building software tools is concerned, it is far too much to hope that one tool or even one tool-kit will apply to all algebras. But every success in finding mathematical coherence among a subfamily of algebras will provide the tool-builders with an opportunity to broaden the use of their software.

References

[1] D. Austry and G. Boudol, *Algèbre de processus et synchronisation*, Journal of Theor. Comp. Science, Vol 30, pp90–131, 1984.

[2] J. Baeten and J. Bergstra, *Real Time Process Algebra*, Formal Aspects of Computing 3(2), pp142-188, 1991.

[3] J. Baeten and J. Bergstra, *Asynchronous Communication in Real Space Process Algebra*, CWI Amsterdam, 1991.

[4] Baeten, J.C.M. and Weijland, W.P., **Process Algebra**, Cambridge University Press 1990.

[5] J. Barnes, *Synchronous CSP*, Oxford 1991.

[6] J. Bradfield and C. Stirling, *Verifying Temporal Properties of Processes*, Proceedings of CONCUR'90, Lecture Notes in Computer Science 458, pp115-125, 1990.

[7] E. Brinksma, **On the design of Extended LOTOS**, A Specification Language for Open Distributed Systems, PhD Thesis, University of Twente, ISBN 90-9002545-6, 1988.

[8] G. Bruns, *A Language for Value-passing CCS*, Edinburgh, 1991.

[9] M. Hennessy, **Algebraic Theory of Processes**, MIT Press, 1988.

[10] M. Hennessy, *A Model for the π-Calculus*, Sussex 1991.

[11] M. Hennessy & A. Ingolfsdottir, *Communicating Processes with Value-passing and Assignments*, Sussex, 1991.

[12] M. Hennessy and T. Regan, *A Process Algebra for Timed Systems*, Sussex 1991.

[13] C.A.R. Hoare, **Communicating Sequential Processes**, Prentice Hall, 1985.

[14] M. Josephs, *Receptive Process Theory*, To appear in Acta Informatica.

[15] G. Milne, *Circal and the representation of communication*, ACM Transactions on Programminig Languages and Systems, Vol 7, pp270–298, 1985.

[16] R. Milner, **Communication and Concurrency**, Prentice Hall, 1989

[17] R. Milner, J. Parrow and D. Walker, *Modal Logics for Mobile Processes*, Invited Lecture, Proceedings of CONCUR'91,

[18] F. Moller and C. Tofts, *Relating Processes with Respect to Speed*, Proceedings of CONCUR'91.

[19] R. de Simone, *Higher-level synchronizing devices in Meije-SCCS*, Journal of Theoretical Computer Science, Vol 37, pp245–267, 1985.