A Brief History of Process Algebra

J.C.M. Baeten

Department of Computer Science, Technische Universiteit Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
josb@win.tue.nl

Abstract. This note addresses the history of process algebra as an area of research in concurrency theory, the theory of parallel and distributed systems in computer science. Origins are traced back to the early seventies of the twentieth century, and developments since that time are sketched. The author gives his personal views on these matters. He also considers the present situation, and states some challenges for the future.
Note: Based on [9]. Also report CS-R 04-02, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, http://www.win.tue.nl/fm/pubbaeten.html.

1 Introduction

Computer science is a very young science, that has its origins in the middle of the twentieth century. For this reason, describing its history, or the history of an area of research in computer science, is an activity that receives scant attention. The organizers of a workshop on process algebra in Bertinoro in July 2003 (see [2]) aimed at describing the current state of affairs and future directions of research in *process algebra*, an active area of research in computer science. In planning the workshop, the question came up how the present situation came about. I was asked to investigate the history of process algebra.

This note addresses the history of process algebra as an area of research in concurrency theory, the theory of parallel and distributed systems in computer science. Origins are traced back to the early seventies of the twentieth century, and developments since that time are sketched.

First of all, I define the term 'process algebra' and consider possible differences of scope that people may have. In the rest of the note, I use a strict definition but also make some remarks about related matters. Then, I sketch the state of research in the early seventies, and state which breakthroughs were needed in order for the theories to appear. I consider the development of CCS, CSP and ACP. In Section 4, I sketch the main developments since then.

Let me stress that I give my personal views on what are the central issues and the important breakthroughs. Others may well have very different views. I welcome discussion, in order to get a better and more complete account. I briefly consider the present situation, and state some challenges for the future.

Acknowledgement

The author gratefully acknowledges the discussions at the Process Algebra: Open Problems and Future Directions Workshop in Bertinoro in July 2003 (see [2]). In particular, I would like to mention input from Jan Friso Groote [44] and Luca Aceto.

2 What is a process algebra?

The term process algebra is used in different meanings. First of all, let me consider the word 'process'. It refers to *behaviour* of a *system*. A system is anything showing behaviour, in particular the execution of a software system, the actions of a machine or even the actions of a human being. Behaviour is the total of events or actions that a system can perform, the order in which they can be executed and maybe other aspects of this execution such as timing or probabilities. Always, we describe certain aspects of behaviour, disregarding other aspects, so we are considering an abstraction or idealization of the 'real' behaviour. Rather, we can say that we have an *observation* of behaviour, and an action is the chosen unit of observation. Usually, the actions are thought to be discrete: occurrence is at some moment in time, and different actions are separated in time. This is why a process is sometimes also called a *discrete event system*.

The word 'algebra' denotes that we take an algebraic/axiomatic approach in talking about behaviour. That is, we use the methods and techniques of universal algebra (see e.g. [61]). In order to make a comparison, consider the definition of a *group* in mathematical algebra.

Definition 1. *A* group *has a signature* $(G, *, u, ^{-1})$ *with the laws or axioms*

- $a * (b * c) = (a * b) * c$
- $u * a = a = a * u$
- $a * a^{-1} = a^{-1} * a = u$

So, a group is any mathematical structure with operators satisfying the group axioms. Stated differently: a group is any model of the equational theory of groups. Likewise, we can say that a process algebra is any mathematical structure satisfying the axioms given for the basic operators. A process is an element of a process algebra. By using the axioms, we can perform *calculations* with processes.

The simplest model of behaviour is to see behaviour as an input/output function. A value or input is given at the beginning of the process, and at some moment there is a(nother) value as outcome or output. This model was used to advantage as the simplest model of the behaviour of a computer program in computer science, from the start of the subject in the middle of the twentieth century. It was instrumental in the development of (finite state) *automata theory*. In automata theory, a process is modeled as an automaton. An automaton has a number of *states* and a number of *transitions*, going from one state to a(nother) state. A transition denotes the execution of an (elementary) action, the basic

unit of behaviour. Besides, there is an initial state (sometimes, more than one) and a number of final states. A behaviour is a run, i.e. a path from initial state to final state. Important from the beginning is when to consider two automata equal, expressed by a notion of equivalence. On automata, the basic notion of equivalence is language equivalence: a behaviour is characterized by the set of executions from the initial state to a final state. An algebra that allows equational reasoning about automata is the algebra of regular expressions (see e.g. [58]).

Later on, this model was found to be lacking in several situations. Basically, what is missing is the notion of *interaction*: during the execution from initial state to final state, a system may interact with another system. This is needed in order to describe parallel or distributed systems, or so-called *reactive* systems. When dealing with interacting systems, we say we are doing *concurrency theory*, so concurrency theory is the theory of interacting, parallel and/or distributed systems. When talking about process algebra, we usually consider it as an approach to concurrency theory, so a process algebra will usually (but not necessarily) have parallel composition as a basic operator.

Thus, we can say that process algebra is the study of the behaviour of parallel or distributed systems by algebraic means. It offers means to describe or *specify* such systems, and thus it has means to talk about parallel composition. Besides this, it can usually also talk about alternative composition (choice) and sequential composition (sequencing). Moreover, we can reason about such systems using algebra, i.e. equational reasoning. By means of this equational reasoning, we can do *verification*, i.e. we can establish that a system satisfies a certain property.

What are these basic laws of process algebra? We can list some, that are usually called *structural* or *static* laws. We start out from a given set of atomic actions, and use the basic operators to compose these into more complicated processes. As basic operators, we use $+$ denoting alternative composition, $;$ denoting sequential composition and $\parallel$ denoting parallel composition. Usually, there are also neutral elements for some or all of these operators, but we do not consider these here. Some basic laws are the following ($+$ binding weakest, $;$ binding strongest).

- $x + y = y + x$ (commutativity of alternative composition)
- $x + (y + z) = (x + y) + z$ (associativity of alternative composition)
- $x + x = x$ (idempotency of alternative composition)
- $(x + y); z = x; z + y; z$ (right distributivity of $+$ over $;$)
- $(x; y); z = x; (y; z)$ (associativity of sequential composition)
- $x \parallel y = y \parallel x$ (commutativity of parallel composition)
- $(x \parallel y) \parallel z = x \parallel (y \parallel z)$ (associativity of parallel composition)

These laws are called static laws because they do not mention action execution explicitly, but only list general properties of the operators involved.

We can see that there is a law connecting alternative and sequential composition. In some cases, other connections are considered. On the other hand, we list no law connecting parallel composition to the other operators. It turns out such a connection is at the heart of process algebra, and it is the tool that makes calculation possible. In most process algebras, this law allows to express parallel

composition in terms of the other operators, and is called an expansion theorem. As this law involves action execution explicitly, it is an example of a so-called *dynamic* law. Process algebras with an expansion theorem are called *interleaving* process algebras, those without are called *partial order* or *true concurrency*. For a discussion concerning this dichotomy, see [8].

So, we can say that any mathematical structure with three binary operations satisfying these 7 laws is a process algebra. Most often, these structures are formulated in terms of automata, in this case mostly called *transition systems*. This means a transition system has a number of states and transitions between them, an initial state and a number of final states. The notion of equivalence studied is usually not language equivalence. Prominent among the equivalences studied is the notion of *bisimulation*. Often, the study of transition systems, ways to define them and equivalences on them are also considered part of process algebra, even in the case no equational theory is present.

In this note, I will be more precise: *process algebra* is the study of pertinent equational theories with their models, while the wider field that also includes the study of transition systems and related structures, ways to define them and equivalences on them will be called *process theory*.

3 History

The history of process algebra will be traced back to the early seventies of the twentieth century. At that point, the only part of concurrency theory that existed is the theory of Petri nets, conceived by Petri starting from his thesis in 1962 [84]. In 1970, we can distinguish three main styles of formal reasoning about computer programs, focusing on giving semantics (meaning) to programming languages.

1. Operational semantics. A computer program is modeled as an execution of an abstract machine. A state of such a machine is a valuation of variables, a transition between states is an elementary program instruction. Pioneer of this field is McCarthy [64].
2. Denotational semantics. More abstract than operational semantics, computer programs are usually modeled by a function transforming input into output. Most well-known are Scott and Strachey [92].
3. Axiomatic semantics. Here, emphasis is put on proof methods proving programs correct. Central notions are program assertions, proof triples consisting of precondition, program statement and postcondition, and invariants. Pioneers are Floyd [38] and Hoare [53].

Then, the question was raised how to give semantics to programs containing a parallel operator. It was found that this is difficult using the methods of denotational, operational or axiomatic semantics, although several attempts are made (for instance, see [50]).

There are two paradigm shifts that need to made, before a theory of parallel programs in terms of a process algebra can be developed. First of all, the idea of a behaviour as an input/output function needed to be abandoned. A program

could still be modeled as an automaton, but the notion of language equivalence is no longer appropriate. This is because the interaction a process has between input and output influences the outcome, disrupting functional behaviour. Secondly, the notion of *global* variables needs to be overcome. Using global variables, a state of a modeling automaton is given as a valuation of the program variables, that is, a state is determined by the values of the variables. The independent execution of parallel processes makes it difficult or impossible to determine the values of global variables at a given moment. It turns out to be simpler to let each process have its own local variables, and to denote exchange of information explicitly.

In the rest of this section, we describe the history of process algebra from the early seventies to the early eighties, by focusing on the central people involved. By the early eighties, we can say process algebra is established as a separate area of research. Section 4 will consider the developments since the early eighties until the present time by subarea.

3.1 Bekič

One of the people studying the semantics of parallel programs in the early seventies was Hans Bekič. He was born in 1936, and died due to a mountain accident in 1982. In the period we are speaking of, he worked at the IBM lab in Vienna, Austria. The lab was well-known in the sixties and seventies for the work on the definition and semantics of programming languages, and Bekič played a part in this, working on the denotational semantics of ALGOL and PL/I. Growing out of his work on PL/I, the problem arose how to give a denotational semantics for parallel composition. Bekič tackled this problem in [18]. This internal report, and indeed all the work of Bekič is made accessible to us through the work of Cliff Jones [19]. On this book, I base the following remarks.

In [18], Bekič addresses the semantics of what he calls "quasi-parallel execution of processes". From the introduction, I quote:

> Our plan to develop an *algebra of processes* may be viewed as a *high-level* approach: we are interested in how to compose complex processes from simpler (still arbitrarily complex) ones.

Bekič uses global variables, so a state ξ is a valuation of variables, and a program determines an action A, which gives, in a state (non-deterministically) either *null* iff it is an end-state, or an elementary step f, giving a new state $f\xi$ and rest-action A'. Further, there are $\sqcup$ and *cases* denoting alternative composition, ; denoting sequential composition, and // denoting (quasi-)parallel composition.

On page 183 in [19], we see the following law for quasi-parallel composition:

$$\begin{array}{l}(A//B)\xi = \\ \quad (\text{cases} \quad A\xi : \text{null} \to B\xi \\ \qquad\qquad\quad (f, A') \to f, (A'//B)) \\ \qquad \sqcup \\ \quad (\text{cases} \quad B\xi : \text{null} \to A\xi \\ \qquad\qquad\quad (g, B') \to g, (A//B'))\end{array}$$

and this is called the "unspecified merging" of the elementary steps of A and B. This is definitely a pre-cursor of what later would be called the expansion law of process algebra. It also makes explicit that Bekič has made the first paradigm shift: the next step in a merge is not determined, so we have abandoned the idea of a program as a function.

The book [19] goes on with clarifications of [18] from a lecture in Amsterdam in 1972. Here, Bekič states that an action is tree-like, behaves like a scheduler, so that for instance $f; (g \sqcup h)$ is not the same as $(f; g) \sqcup (f; h)$ for elementary steps f, g, h, another example of non-functional behaviour. In a letter to Peter Lucas from 1975, Bekič is still struggling with his notion of an action, and writes:

> These actions still contain enough information so that the normal operations can be defined between them, but on the other hand little enough information to fulfil certain desirable equivalences, such as:
>
> $$a; 0 = a \quad a; (b; c) = (a; b); c \quad a//b = b//a$$
>
> etc.

In a lecture on the material in 1974 in Newcastle, Bekič has changed the notation of // to $\|$ and calls the operator parallel composition. In giving the equations, we even encounter a "left-parallel" operator, with laws, with the same meaning that Bergstra and Klop will later give to their left-merge operator [20].

Concluding, we can say that Bekič contributed a number of basic ingredients to the emergence of process algebra, but we see no coherent comprehensive theory yet.

3.2 CCS

The central person in the history of process algebra without a doubt is Robin Milner. A.J.R.G. Milner, born in 1934, developed his process theory CCS (Calculus of Communicating Systems) over the years 1973 to 1980, culminating in the publication of the book [72] in 1980.

The oldest publications concerning the semantics of parallel composition are [67, 68], formulated within the framework of denotational semantics, using so-called transducers. He considers the problems caused by non-terminating programs, with side effects, and non-determinism. He uses the operations $*$ for

sequential composition, ? for alternative composition and $\|$ for parallel composition. He refers to [18] as related work.

Next, chronologically, are the articles [71, 66]. Here, Milner introduces *flow graphs*, with ports where a named port synchronizes with the port with its co-name. Operators are $|$ for parallel composition, restriction and relabeling. The symbol $\|$ is now reserved for restricted parallel composition. Static laws are stated for these operators.

The following two papers are [69, 70], putting in place most of CCS as we know it. The dynamic operators prefixing and alternative composition are added and provided with laws. Synchronization trees are used as a model. The prefix τ occurs as a *communication trace* (what remains of a synchronization of a name and a co-name). The paradigm of message passing is taken over from [54]. Interleaving is introduced as the observation of a single observer of a communicating system, and the expansion law is stated. Sequential composition is not a basic operator, but a derived one, using communication, abstraction and restriction.

The paper [49], with Matthew Hennessy, formulates basic CCS, with observational equivalence and strong equivalence defined inductively. Also, so-called Hennessy-Milner logic is introduced, which provides a logical characterization of process equivalence. Next, the book [72] is the standard process algebra reference. Here we have for the first time in history a complete process algebra, with a set of equations and a semantical model. In fact, Milner talks about *process calculus* everywhere in his work, emphasizing the calculational aspect. He presents the equational laws as truths about his chosen semantical domain, rather than considering the laws as primary, and investigating the range of models that they have.

An important contribution that was realized just after the appearance of [72] is the formulation of bisimulation by David Park [83]. This became a central notion in process theory subsequently. The book [72] was later updated in [75].

A related development is the birth of *structural operational semantics* in [86]. More can be read about this in the recent history paper [87].

3.3 CSP

A very important contributor to the development of process algebra is Tony Hoare. C.A.R. Hoare, born in 1934, published the influential paper [54] as a technical report in 1976. The important step is that he does away completely with global variables, and adopts the message passing paradigm of communication, thus realizing the second paragdigm shift. The language CSP (Communicating Sequential Processes) described in [54] has synchronous communication and is a guarded command language (based on [35]). No model or semantics is provided. This paper inspired Milner to treat message passing in CCS in the same way.

A model for CSP was elaborated in [55]. This is a model based on trace theory, i.e. on the sequences of actions a process can perform. Later on, it was found that this model was lacking, for instance because deadlock behaviour is not preserved. For this reason, a new model based on failure pairs was presented in [29] for the language that was then called TCSP (*Theoretical CSP*). Later,

TCSP was called CSP again. Some time later it was established that the failure model is the least discriminating model that preserves deadlock behaviour. In the language, due to the presence of two alternative composition operators, it is possible to do without a silent step like τ altogether. The book [56] gives a good overview of CSP.

3.4 Some Other Process Theories

Around 1980, concurrency theory and in particular process theory is a vibrant field with a lot of activity world wide. We already mentioned research on Petri nets, that is an active area [85]. Another partial order process theory is given in [63]. Research on temporal logic has started, see e.g. [88].

Some other process theories can be mentioned. We already remarked that Hoare investigated trace theory. More work was done in this direction, e.g. by Rem [90]. There is also the invariants calculus [6].

Another process theory that should be mentioned is the metric approach by De Bakker and Zucker [16, 17]. There is a notion of distance between processes: processes that do not differ in behaviour before the nth step have a distance of at most 2^{-n}. This turns the domain of processes into a metric space, that can be completed, and solutions to guarded recursive equations exist by application of Banach's fixed point theorem.

3.5 ACP

Jan Bergstra and Jan Willem Klop in 1982 started work on a question of De Bakker as to what can be said about solutions of unguarded recursive equations. As a result, they wrote the paper [20]. In this paper, the phrase "process algebra" is used for the first time. We quote:

> A *process algebra* over a set of atomic actions A is a structure $\mathcal{A} = \langle \mathsf{A}, +, \cdot, \between, a_i (i \in I) \rangle$ where A is a set containing A, the a_i are constant symbols corresponding to the $a_i \in A$, and $+$ (*union*), $\cdot$ (*concatenation* or *composition*, left out in the axioms), $\between$ (*left merge*) satisfy for all $x, y, z \in \mathsf{A}$ and $a \in A$ the following axioms:

PA1	$x + y = y + x$
PA2	$x + (y + z) = (x + y) + z$
PA3	$x + x = x$
PA4	$(xy)z = x(yz)$
PA5	$(x + y)z = xz + yz$
PA6	$(x + y) \between z = x \between z + y \between z$
PA7	$ax \between y = a(x \between y + y \between x)$
PA8	$a \between y = ay$

This clearly establishes process algebra in the strict sense as I set it out in Section 2. In the paper, process algebra was defined with alternative, sequential and parallel composition, but without communication. A model was established based on projective sequences (a process is given by a sequence of approximations by finite terms), and in this model, it is established that all recursive equations have a solution. In adapted form, this paper was later published as [23]. In [22], this process algebra PA was extended with communication to yield the theory ACP (Algebra of Communicating Processes). The book [15] gives an overview of ACP.

Comparing the three most well-known process algebras CCS, CSP and ACP, we can say there is a considerable amount of work and applications realized in all three of them. Historically, CCS was the first with a complete theory. Different from the other two, CSP has a least distinguishing equational theory. More than the other two, ACP emphasizes the algebraic aspect: there is an equational theory with a range of semantical models. Also, ACP has a more general communication scheme: in CCS, communication is combined with abstraction, in CSP, communication is combined with restriction.

3.6 Further Remarks

In later years, other process algebras were developed. We can mention SCCS [73], CIRCAL [65], MEIJE [7], the process algebra of Hennessy [48].

We see that over the years many process algebras have been developed, each making its own set of choices in the different possibilities. The reader may wonder whether this is something to be lamented. In the paper [11], it is argued that this is actually a good thing, as long as there is a good exchange of information between the different groups, as each different process algebra will have its own set of advantages and disadvantages. I do think it is lamentable that the community still has not been able to come up with a uniform notation, denoting the same thing in the same way.

4 Developments

In this section, I will address a number of important developments that have occurred since the formulation of the basic process algebras CCS, CSP and ACP.

4.1 Theory

A nice overview of the most important theoretical results since the start of process algebra is the recent paper [1]. I already mentioned the formulation of the notion of bisimulation by Park [83]. Strong bisimulation and weak bisimulation became the central notion of equivalence in process theory. As an alternative to weak bisimulation, *branching bisimulation* was proposed in [42] with certain advantages [40]. In between bisimulation and trace equivalence there is a whole

lattice of other equivalences, see the overview in [41]. See also [39]. For process algebra based on partial order semantics, see [27].

There is a wealth of results concerning process algebra extended with some form of recursion, see e.g. the complete axiomatization of regular processes by Milner [74] or the overview on decidability in [30].

Structural operational semantics (SOS) has become the dominant way of providing a model for a process algebra. It is standard practice to provide a new operation with SOS rules, even before an equational characterization is attempted. There are many results based on the *formats* of these rules. An overview can be found in [3].

Finally, there is a whole range of *expressiveness* results, some examples can be found in [21].

As is also stated in [1], there are many nice results, but also many remaining open problems.

4.2 Tooling

Over the years, several software systems have been developed in order to facilitate the application of process algebra in the analysis of systems. In this subsection, I only mention general process algebra tools. Tools that deal with specific extensions are mentioned below.

The most well-known general tool is the Concurrency Workbench, see [80], dealing with CCS type process algebra. There is also the variant CWB-NC, see [98] for the current state of affairs. There is the French set of tools CADP, see e.g. [37]. In the CSP tradition, there is the FDR tool (see `http://www.fsel.com/`).

The challenge in tool development is to combine an attractive user interface with a powerful and fast back engine.

4.3 Verification

A measure of success of process algebra is the systems that have been successfully verified by means of techniques that come from process algebra. A good overview of the current state of affairs is [46]. Process algebra focuses on equational reasoning. Other successful techniques are model checking and theorem proving. Combination of these different approaches proves to be very promising.

4.4 Data

Process algebra is very successful in describing the dynamic behaviour of systems. In describing the static aspects, treatment of data is very important. Actions and processes are parametrized with data elements. The combination of processes and data has received much attention over the years. A standardized formal description technique is LOTOS, see [28]. Another combination of processes and data is PSF, see [62] with associated tooling. The process algebra with data μCRL has tooling focusing on equational verification, see e.g. [45].

In model checking, so-called *symbolic* model checking is employed in order to deal with data parameters.

4.5 Time

Research on process algebra extended with a quantitative notion of time started with the work of Reed and Roscoe in the CSP context, see [89]. A textbook in this tradition is [91].

There are many variants of CCS with timing, see e.g. [96], [81].

In the ACP tradition, work starts with [10]. An integrated theory, involving both discrete and dense time, both relative and absolute time, is presented in the book [13].

Also the theory ATP can be mentioned, see [82].

There are many formulations, in these works and elsewhere, of bisimulation that takes into account timing information. Mostly, the passage of time is treated as the occurrence of an action: bisimilar processes must have exactly matching timing behaviour. Recent protocol verification has shown that this may be too strict: perhaps a less discriminating equivalence should be used with respect to timing information (see [14]). More research is needed in this direction. Also a notion of *approximation* would be very useful. More experience with verification of functional behaviour and timing behaviour is certainly needed.

Tooling has been developed for processes with timing mostly in terms of timed automata, see e.g. UPPAAL [57] or KRONOS [97]. Equational reasoning is investigated for μCRL with timing [93].

4.6 Mobility

Research on networks of processes where processes are mobile and configuration of communication links is dynamic has been dominated by the π-calculus. An early reference is [36], the standard reference is [79] and the textbook is [77]. The associated tool is the Mobility Workbench, see [94]. Also in this domain, it is important to gain more experience with protocol verification. On the theory side, there are a number of different equivalences that have been defined, and it is not clear which is the 'right' one to use.

More recently, other calculi concerning mobility have been developed, notably the *ambient calculus*, see [31]. As to unifying frameworks for different mobile calculi, Milner investigates action calculus [76] and bigraphs [78].

4.7 Probabilities and Stochastics

Process algebras extended with probabilistic or stochastic information have generated a lot of research in recent years. An early reference is [47]. In the CSP tradition, there is [59], in the CCS tradition [52], in the ACP tradition [12]. There is the process algebra TIPP with associated tool, see e.g. [43], and EMPA, see e.g. [26].

Recently, the insight that both (unquantified) alternative composition and probabilistic choice are needed for a useful theory has gained attention, see e.g. the work in [33] or [5].

Notions of abstraction are still a matter of continued research. The goal is to combine functional verification with performance analysis. A notion of approximation is very important here, for an attempt see [34]. Much further work needs to be done on probabilistic and stochastic process algebra.

4.8 Hybrid Systems

Systems that in their behaviour depend on continuously changing variables other than time are the latest challenge to be addressed by process algebra. System descriptions involve differential algebraic equations, connections with dynamic control theory are very important. Here, process algebra research is just at the beginning. The attempts [24, 32] are worth mentioning.

In process theory, work centres around *hybrid automata* [4] and *hybrid I/O automata* [60]. A tool is HyTech, see [51]. A connection with process algebra can be found in [95].

5 Conclusion

In this note, a brief history is sketched of process algebra. The early work centred around giving semantics to programming languages involving a parallel construct. Here, two breakthroughs were needed: first of all, abandoning the idea that a program is a transformation from input to output, replacing this by an approach where all intermediate states are important, and, secondly, replacing the notion of global variables by the paradigm of message passing and local variables.

In the seventies of the twentieth century, both these steps were taken, and the process algebras CCS and CSP evolved. In doing so, process algebra became an underlying theory of all parallel and distributed systems, extending formal language and automata theory with the central ingredient of *interaction*.

In the following years, much work has been done, and many process algebras have been formulated, extended with data, time, mobility, probabilities and stochastics. The work is not finished, however. I formulated some challenges for the future. More can be found in [2].

References

1. L. Aceto. Some of my favorite results in classic process algebra. Technical Report NS-03-2, BRICS, 2003.
2. L. Aceto, Z. Ésik, W.J. Fokkink, and A. Ingólfsdóttir, editors. *Process Algebra: Open Problems and Future Directions*. BRICS Notes Series NS-03-3, 2003.
3. L. Aceto, W.J. Fokkink, and C. Verhoef. Structural operational semantics. In [25], pp. 197–292, 2001.
4. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

5. S. Andova. *Probabilistic Process Algebra*. PhD thesis, Technische Universiteit Eindhoven, 2002.
6. K.R. Apt, N. Francez, and W.P. de Roever. A proof system for communicating sequential processes. *TOPLAS*, 2:359–385, 1980.
7. D. Austry and G. Boudol. Algèbre de processus et synchronisation. *Theoretical Computer Science*, 30:91–131, 1984.
8. J.C.M. Baeten. The total order assumption. In S. Purushothaman and A. Zwarico, editors, *Proceedings First North American Process Algebra Workshop*, Workshops in Computing, pages 231–240. Springer Verlag, 1993.
9. J.C.M. Baeten. Over 30 years of process algebra: Past, present and future. In L. Aceto, Z. Ésik, W.J. Fokkink, and A. Ingólfsdóttir, editors, *Process Algebra: Open Problems and Future Directions*, volume NS-03-3 of *BRICS Notes Series*, pages 7–12, 2003.
10. J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.
11. J.C.M. Baeten, J.A. Bergstra, C.A.R. Hoare, R. Milner, J. Parrow, and R. de Simone. The variety of process algebra. Deliverable ESPRIT Basic Research Action 3006, CONCUR, 1991.
12. J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 121(2):234–255, 1995.
13. J.C.M. Baeten and C.A. Middelburg. *Process Algebra with Timing*. EATCS Monographs. Springer Verlag, 2002.
14. J.C.M. Baeten, C.A. Middelburg, and M.A. Reniers. A new equivalence for processes with timing. Technical Report CSR 02-10, Eindhoven University of Technology, Computer Science Department, 2002.
15. J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.
16. J.W. de Bakker and J.I. Zucker. Denotational semantics of concurrency. In *Proceedings 14th Symposium on Theory of Computing*, pages 153–158. ACM, 1982.
17. J.W. de Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54:70–120, 1982.
18. H. Bekič. Towards a mathematical theory of processes. Technical Report TR 25.125, IBM Laboratory Vienna, 1971.
19. H. Bekič. *Programming Languages and Their Definition (Selected Papers edited by C.B. Jones)*. Number 177 in LNCS. Springer Verlag, 1984.
20. J.A. Bergstra and J.W. Klop. Fixed point semantics in process algebra. Technical Report IW 208, Mathematical Centre, Amsterdam, 1982.
21. J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proceedings 11th ICALP*, number 172 in LNCS, pages 82–95. Springer Verlag, 1984.
22. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.
23. J.A. Bergstra and J.W. Klop. A convergence theorem in process algebra. In J.W. de Bakker and J.J.M.M. Rutten, editors, *Ten Years of Concurrency Semantics*, pages 164–195. World Scientific, 1992.
24. J.A. Bergstra and C.A. Middelburg. Process algebra semantics for hybrid systems. Technical Report CS-R 03/06, Technische Universiteit Eindhoven, Dept. of Comp. Sci., 2003.
25. J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. North-Holland, Amsterdam, 2001.

26. M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with non-determinism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.
27. E. Best, R. Devillers, and M. Koutny. A unified model for nets and process algebras. In [25], pp. 945–1045, 2001.
28. E. Brinksma, editor. *Information Processing Systems, Open Systems Interconnection, LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, volume IS-8807 of *International Standard*. ISO, Geneva, 1989.
29. S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.
30. O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In [25], pp. 545–623, 2001.
31. L. Cardelli and A.D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, 2000.
32. P.J.L. Cuijpers and M.A. Reniers. Hybrid process algebra. Technical Report CS-R 03/07, Technische Universiteit Eindhoven, Dept. of Comp. Sci., 2003.
33. P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. PhD thesis, University of Twente, 1999.
34. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labeled Markov systems. In J.C.M. Baeten and S. Mauw, editors, *Proceedings CONCUR'99*, number 1664 in Lecture Notes in Computer Science, pages 258–273. Springer Verlag, 1999.
35. E.W. Dijkstra. Guarded commands, nondeterminacy, and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.
36. U. Engberg and M. Nielsen. A calculus of communicating systems with label passing. Technical Report DAIMI PB-208, Aarhus University, 1986.
37. J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP (CAESAR/ALDEBARAN development package): A protocol validation and verification toolbox. In R. Alur and T.A. Henzinger, editors, *Proceedings CAV '96*, number 1102 in Lecture Notes in Computer Science, pages 437–440. Springer Verlag, 1996.
38. R.W. Floyd. Assigning meanings to programs. In J.T. Schwartz, editor, *Proceedings Symposium in Applied Mathematics*, Mathematical Aspects of Computer Science, pages 19–32. AMS, 1967.
39. R.J. van Glabbeek. The linear time – branching time spectrum II; the semantics of sequential systems with silent moves. In E. Best, editor, *Proceedings CONCUR '93*, number 715 in Lecture Notes in Computer Science, pages 66–81. Springer Verlag, 1993.
40. R.J. van Glabbeek. What is branching time semantics and why to use it? In M. Nielsen, editor, *The Concurrency Column*, pages 190–198. *Bulletin of the EATCS* 53, 1994.
41. R.J. van Glabbeek. The linear time – branching time spectrum I. The semantics of concrete, sequential processes. In [25], pp. 3–100, 2001.
42. R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43:555–600, 1996.
43. N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In L. Donatiello and R. Nelson, editors, *Performance Evaluation of Computer and Communication Systems*, number 729 in LNCS, pages 121–146. Springer, 1993.

44. J.F. Groote. *Process Algebra and Structured Operational Semantics*. PhD thesis, University of Amsterdam, 1991.
45. J.F. Groote and B. Lisser. Computer assisted manipulation of algebraic process specifications. Technical Report SEN-R0117, CWI, Amsterdam, 2001.
46. J.F. Groote and M.A. Reniers. Algebraic process verification. In [25], pp. 1151–1208, 2001.
47. H. Hansson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, University of Uppsala, 1991.
48. M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
49. M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In J.W. de Bakker and J. van Leeuwen, editors, *Proceedings 7th ICALP*, number 85 in Lecture Notes in Computer Science, pages 299–309. Springer Verlag, 1980.
50. M. Hennessy and G.D. Plotkin. Full abstraction for a simple parallel programming language. In J. Becvar, editor, *Proceedings MFCS*, number 74 in LNCS, pages 108–120. Springer Verlag, 1979.
51. T.A. Henzinger, P. Ho, and H. Wong-Toi. Hy-Tech: The next generation. In *Proceedings RTSS*, pages 56–65. IEEE, 1995.
52. J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, Cambridge University Press, 1996.
53. C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969.
54. C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.
55. C.A.R. Hoare. A model for communicating sequential processes. In R.M. McKeag and A.M. Macnaghten, editors, *On the Construction of Programs*, pages 229–254. Cambridge University Press, 1980.
56. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
57. K.G. Larsen, P. Pettersson, and Wang Yi. Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer*, 1, 1997.
58. P. Linz. *An Introduction to Formal Languages and Automata*. Jones and Bartlett, 2001.
59. G. Lowe. *Probabilities and Priorities in Timed CSP*. PhD thesis, University of Oxford, 1993.
60. N. Lynch, R. Segala, F. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In T. Henzinger, R. Alur, and E. Sontag, editors, *Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science. Springer Verlag, 1995.
61. S. MacLane and G. Birkhoff. *Algebra*. MacMillan, 1967.
62. S. Mauw. *PSF: a Process Specification Formalism*. PhD thesis, University of Amsterdam, 1991. See `http://carol.science.uva.nl/~psf/`.
63. A. Mazurkiewicz. Concurrent program schemes and their interpretations. Technical Report DAIMI PB-78, Aarhus University, 1977.
64. J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirshberg, editors, *Computer Programming and Formal Systems*, pages 33–70. North-Holland, Amsterdam, 1963.
65. G.J. Milne. CIRCAL: A calculus for circuit description. *Integration*, 1:121–160, 1983.
66. G.J. Milne and R. Milner. Concurrent processes and their syntax. *Journal of the ACM*, 26(2):302–321, 1979.
67. R. Milner. An approach to the semantics of parallel programs. In *Proceedings Convegno di informatica Teoretica*, pages 285–301, Pisa, 1973. Instituto di Elaborazione della Informazione.

68. R. Milner. Processes: A mathematical model of computing agents. In H.E. Rose and J.C. Shepherdson, editors, *Proceedings Logic Colloquium*, number 80 in Studies in Logic and the Foundations of Mathematics, pages 157–174. North-Holland, 1975.
69. R. Milner. Algebras for communicating systems. In *Proc. AFCET/SMF joint colloquium in Applied Mathematics*, Paris, 1978.
70. R. Milner. Synthesis of communicating behaviour. In J. Winkowski, editor, *Proc. 7th MFCS*, number 64 in LNCS, pages 71–83, Zakopane, 1978. Springer Verlag.
71. R. Milner. Flowgraphs and flow algebras. *Journal of the ACM*, 26(4):794–818, 1979.
72. R. Milner. *A Calculus of Communicating Systems.* Number 92 in Lecture Notes in Computer Science. Springer Verlag, 1980.
73. R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.
74. R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer System Science*, 28:439–466, 1984.
75. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
76. R. Milner. Calculi for interaction. *Acta Informatica*, 33:707–737, 1996.
77. R. Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, 1999.
78. R. Milner. Bigraphical reactive systems. In K.G. Larsen and M. Nielsen, editors, *Proceedings CONCUR '01*, number 2154 in LNCS, pages 16–35. Springer Verlag, 2001.
79. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100:1–77, 1992.
80. F. Moller and P. Stevens. Edinburgh Concurrency Workbench user manual (version 7.1). Available from `http://www.dcs.ed.ac.uk/home/cwb/`.
81. F. Moller and C. Tofts. A temporal calculus of communicating systems. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR'90*, number 458 in LNCS, pages 401–415. Springer Verlag, 1990.
82. X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.
83. D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings 5th GI Conference*, number 104 in LNCS, pages 167–183. Springer Verlag, 1981.
84. C.A. Petri. *Kommunikation mit Automaten.* PhD thesis, Institut fuer Instrumentelle Mathematik, Bonn, 1962.
85. C.A. Petri. Introduction to general net theory. In W. Brauer, editor, *Proc. Advanced Course on General Net Theory, Processes and Systems*, number 84 in LNCS, pages 1–20. Springer Verlag, 1980.
86. G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.
87. G.D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming*, To appear, 2004. Special Issue on Structural Operational Semantics.
88. A. Pnueli. The temporal logic of programs. In *Proceedings 19th Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.
89. G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.
90. M. Rem. Partially ordered computations, with applications to VLSI design. In J.W. de Bakker and J. van Leeuwen, editors, *Foundations of Computer Science*

IV, volume 159 of *Mathematical Centre Tracts*, pages 1–44. Mathematical Centre, Amsterdam, 1983.
91. S.A. Schneider. *Concurrent and Real-Time Systems (the CSP Approach)*. Worldwide Series in Computer Science. Wiley, 2000.
92. D.S. Scott and C. Strachey. Towards a mathematical semantics for computer languages. In J. Fox, editor, *Proceedings Symposium Computers and Automata*, pages 19–46. Polytechnic Institute of Brooklyn Press, 1971.
93. Y.S. Usenko. *Linearization in μCRL*. PhD thesis, Technische Universiteit Eindhoven, 2002.
94. B. Victor. *A Verification Tool for the Polyadic π-Calculus*. Licentiate thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994. Available as report DoCS 94/50.
95. T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. PhD thesis, Technische Universiteit Eindhoven, 2003.
96. Wang Yi. Real-time behaviour of asynchronous agents. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR'90*, number 458 in LNCS, pages 502–520. Springer Verlag, 1990.
97. S. Yovine. Kronos: A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1:123–133, 1997.
98. D. Zhang, R. Cleaveland, and E. Stark. The integrated CWB-NC/PIOAtool for functional verification and performance analysis of concurrent systems. In H. Garavel and J. Hatcliff, editors, *Proceedings TACAS '03*, number 2619 in Lecture Notes in Computer Science, pages 431–436. Springer-Verlag, 2003.