Book Reviews

J. A. BERGSTRA, A. PONSE AND S. A. SMOLKA (editors) *Handbook of Process Algebra.* Elsevier Science Publishers, 2001, 1356 pp hardbound, ISBN 0-444-82830-3.

Process algebra is a standard subject in Theoretical Computer Science, one that can be introduced to undergraduates in their first year and applied in projects in their final year. There is a massive amount of theory, some acceptable software tools and a decent tradition of applications and case studies. Process algebra is also being used to solve computational problems in several industries.

It is more than 20 years since the publication of Robin Milner's *A Calculus of Communicating Systems* (1980 *Springer Lecture Notes in Computer Science*, **92**), which marks the beginning of modern process theory. Inspired by the lambda calculus, Milner presented a calculus called CCS (Calculus of Communicating Systems) that led to new ways of thinking about the theory of concurrent systems. For example, in the early 1980s, Tony Hoare (along with his students Stephen Brookes and Bill Roscoe) abstracted the calculus TCSP (Theoretical Communicating Sequential Processes) from his ideas for a parallel language CSP of 1978. Jaco de Bakker and Jeff Zucker published a theory of parallel processes based on metric space methods in 1982, an inspiring combination of Milner's view of processes with Maurice Nivat's metric space methods for formal languages containing infinite strings. Jan Bergstra and Jan Willem Klop published their algebraic axiomatization ACP (Algebra of Communicating Processes) in 1982, working under the influence of both the CCS and metric space approaches. Bergstra coined the term 'process algebra' and, at the time, it denoted a technical distinction between the approaches. Roughly speaking, CCS was a calculus designed with an operational semantics in mind, TCSP was about a fixed semantic model (the failures model) and ACP was a purely axiomatic and algebraic theory.

Today the subject of process algebra is the analysis and application of *many* different process algebras and calculi, which have evolved from theories such as CCS, TCSP and ACP. Through the struggle for survival, in the hostile competitive environments of scientific research and of the world's work, they have multiplied and matured. The different process algebras are designed to express different fundamental intuitions about concurrent processes or to solve different system modelling problems.

What is a process algebra? Typically, a process algebra possesses a set of operators for composing processes in various ways. Thus, a system is described by a system of equations or even an algebraic term over the set of operators. These operators are often axiomatized by sets of laws that define their effects. A process algebra has several equivalences and orderings that express the similarity or refinement of terms representing system behaviour and also an operational semantics that enables terms to be mapped into labelled transition systems. By calculating with the laws or unfolding operational definitions, one may reason that one concurrent system description (e.g., a specification) is equivalent to another (e.g., an implementation).

One can approach the subject either as a fundamental study of computational processes enjoying communication and concurrency or as a set of mathematical techniques for modelling systems, giving semantics to system description languages, and, in particular, reasoning about systems.

The *Handbook of Process Algebra* aims to review the current state of the art. The Handbook is large—much larger than a review such as this can possibly do justice to. The 1300+ pages of the Handbook are divided into 19 chapters spread over the following six parts:

(1) Basic theory,
(2) Finite state processes,
(3) Infinite state processes,
(4) Extensions to processes,
(5) Non-interleaving process algebras,
(6) Tools and applications.

Each part contains excellent chapters written by 34 experts who have helped shape their chosen areas. What is covered?

Basic theory has some splendid chapters. To make sense of the diversity of process algebras remains a problem for researchers and students. Surely it is possible to develop *some* general theory of process description languages? Two chapters make an attempt: van Glabbeek's is a major review of process semantics (being an extension of his notable 1990 report), and the chapter of Aceto, Fokkink and Verhoef on structural operational semantics uses general rule formats which allow general properties to be derived. In addition, there are introductions to the fundamental topics of trace semantics, by Olderog and Broy, and to modal and temporal logics and mu-calculi, by Bradfield and Stirling.

Finite state processes has a study of iteration and recursive operations (primarily the Kleene *) by Bergstra, Fokkink and Ponse, and a chapter on algorithms for checking equivalence and refinement for finite-state labelled transition systems by Cleaveland and Sokolsky.

Infinite state processes has a major review of equivalence checking and model checking for infinite-state processes by Burkart, Caucal, Moller and Steffen. There are studies of two basic forms of value passing between processes: for data, as can be found in a multitude of computations, by Ingólfsdóttir and Lin, and for communication connections, as is possible in mobile systems, by Parrow.

Extensions to processes begins with a chapter on adding time, by Baeten and Middelburg. Impressively, the authors deal thoroughly with both discrete and continuous time, the 'embedding' of untimed process algebras in these

timed extensions and the relation between discrete and continuous time process algebras. There is a short chapter by Jonsson, Yi and Larsen on the very difficult topic of adding probabilities to process algebras. The last extension considered allows priorities inside processes, in a chapter by Cleaveland, Lüttgen and Natarajan.

Process algebras were largely built on the idea that concurrent or parallel execution was modelled by the interleaving of actions. Specifically there were expansion theorems in which parallel operators were expressed or eliminated, using choice and sequential operators. Indeed, in the beginning, interleaving was one of the few things on which researchers could agree. This penultimate part of the Handbook discards this limitation. *Non-interleaving process algebras* contains chapters in which the modelling of networks and their concurrent behaviour play an essential role. The first, by Baeten and Basten, concerns the connection between partial order theories of concurrency, including Petri net models, and process algebra. The second, by Best, Devillers and Koutny, considers general nets and equips them with a Petri net semantics and a process algebra semantics which are shown to be equivalent. A chapter by Castellani reviews the role of locality in systems. The last chapter by Gorrieri and Rensink is a survey of research on the refinement of atomic actions necessary for a hierarchical theory of the top-down development of systems.

The final part on *Tools and applications* contains three chapters. The first, by Groote and Reniers, is devoted to the practical verification of distributed algorithms and protocols. The second, by Bergstra, Middelburg and Usenko, adds a range of constructs to discrete time process algebra in order to give a formal semantics for a simplified version of SDL, a language used in telecommunications. Lastly, the chapter by Mauw and Reniers gives a semantical theory for a graphical language for requirements specification and design developed by Philips.

With such tremendous coverage and high-quality chapters, my own *desiderata* are minor and reflect my tastes rather than shortcomings of the work. For example, I regret readers will not develop a better understanding of the origins of the subject. More could have been made of connections with formal language theory, the topological theory of processes, lambda calculus, verification of parallel programs and operating system techniques, which were and remain influential in clarifying notions of process.

The origins are also the key to understanding the diversity of process algebras. A decade ago most of the senior researchers in process algebra were partners in the ESPRIT Basic Research Action 3006, called Concur—Theories of Concurrency: Unification and Extension. Concur must rank among the most influential ESPRIT projects, with the highest Bang for Buck. It led to a new understanding, along with a huge amount of competition, published deliverables and an important colloquium that continues to this day. This Handbook is one of its legacies too, for the people who belonged to Concur authored most of its pages. Each year, at the Annual Review, the supporters of the different approaches came to market with their new features and case studies. The Commissions' reviewers (of which I was one) marvelled at the extensions and applications but sought in vain for the unifications. In a paper called *The Variety of Process Algebras*, signed by the senior scientists from six of the participating sites, the question of unification was discussed carefully. It argued that, whilst some process algebras can be encoded in others, they are not there to be unified. Simply, there are many process algebras and they need their own theories, just as there are many classical algebraic systems (groups, rings, fields, etc.) that are intimately related but exist independently, having rich theories and special areas of application. A revision of this paper might have made an interesting introductory chapter in the Handbook.

Nowadays, applications are many and varied and there are many case studies to choose from. There is some splendid work on modelling physical and biological systems, on hardware and on safety critical computing, including languages for reasoning about railways, that might have received some attention. Once seen, these particular applications of process algebra are not easily forgotten. Finally, after 1300+ pages of process algebra, the algebraist in me wonders whatever happened to the homomorphisms? Will they ever make themselves felt in the subject? Perhaps that brings us back to the unification problem again.

To learn process algebra one needs to study some basic texts on one of the main systems, for example: Milner's *Communication and Concurrency* (Prentice-Hall, 1989); Hoare's *Communicating Sequential Processes* (Prentice-Hall, 1985); Hennessy's *Algebraic Theory of Processes* (MIT Press, 1988); Baeten and Weijland's *Process Algebra* (Cambridge University Press, 1990); or, most recently, Fokkink's lovely *Introduction to Process Algebra* (Springer, 2000).

For the computer scientist interested in fundamental theory or in system design, the Handbook is a treasure house of insights and information. It is an essential reference work for all academic and Research and Development libraries and will repay handsomely hours spent browsing and studying some of its excellent chapters. It is not easy to find authors with the knowledge, time *and* stamina to deliver high-quality chapters. The editors must be congratulated in having created a major landmark in this exciting area of science.

And exciting it really is! We should recognize that process algebra is a real achievement of Computer Science. It is a subject that computer scientists invented to solve their own fundamental problems. It is a subject with huge potential. It is making its mark in the pop culture of design technologies, of course. However, I think it will grow in its intellectual influence and will secure the glittering prizes of applications in science. But before it makes its mark in one of the classical cultures of physics or biology, its theoretical development has a long way to go.

J. V. TUCKER
University of Wales Swansea