ELSEVIER SCIENCE B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

First edition 2001

Library of Congress Cataloging in Publication Data
A catalog record from the Library of Congress has been applied for.

ISBN: 0-444-82830-3

♾ The paper used in this publication meets the requirements of ANSI/NISO Z39.48-1992 (Permanence of Paper).

Printed in The Netherlands

# Preface

## 1. Introduction

According to the Oxford English Dictionary (OED II CD-ROM), a *process* is a series of actions or events, and an *algebra* is a calculus of symbols combining according to certain defined laws. Completing the picture, a *calculus* is a system or method of calculation. Despite going back as far as the 13th Century, collectively, these definitions do a good job of accurately conveying the meaning of this Handbook's subject: *process algebra.*

A process algebra is a formal description technique for complex computer systems, especially those with communicating, concurrently executing components. A number of different process algebras have been developed – ACP [1], CCS [6], and TCSP [2] being perhaps the best-known – but all share the following key ingredients.

- **Compositional modeling.** Process algebras provide a small number of constructs for building larger systems up from smaller ones. CCS, for example, contains six operators in total, including ones for composing systems in parallel and others for choice and scoping.
- **Operational semantics.** Process algebras are typically equipped with a Plotkin-style [7] structural operational semantics (SOS) that describes the single-step execution capabilities of systems. Using SOS, systems represented as terms in the algebra can be "compiled" into labeled transition systems.
- **Behavioral reasoning via equivalences and preorders.** Process algebras also feature the use of behavioral relations as a means for relating different systems given in the algebra. These relations are usually equivalences, which capture a notion of "same behavior", or preorders, which capture notions of "refinement".

In a process-algebraic approach to system verification, one typically writes two specifications. One, call it SYS, captures the design of the actual system and the other, call it SPEC, describes the system's desired "high-level" behavior. One may then establish the correctness of SYS with respect to SPEC by showing that SYS behaves the "same as" SPEC (if using an equivalence) or by showing that it refines SPEC (if using a preorder).

Establishing the correctness of SYS with respect to SPEC can be done in a syntax-oriented manner or in a semantics-oriented manner. In the former case, an *axiomatization* of the behavioral relation of choice is used to show that one expression can be transformed into the other via syntactic manipulations. In the latter case, one can appeal directly to the definition of the behavioral relation, and to the operational semantics of the two expressions, to show that they are related. In certain cases, e.g., when SYS and SPEC are "finite-state", verification, be it syntax-based or semantics-based, can be carried out automatically.

The advantages to an algebraic approach are the following.

- **System designers need learn only one language** for specifications and designs.
- **Related processes may be substituted for one another** inside other processes. This makes process algebras particularly suitable for the *modular analysis* of complex systems, since a specification and a design adhering to this specification may be used interchangeably inside larger systems.
- **Processes may be minimized** with respect to the equivalence relation before being analyzed; this sometimes leads to orders of magnitude improvement in the performance of verification routines.

Process-algebraic system descriptions can also be verified using *model checking* [3], a technique for ascertaining if a labeled transition system satisfies a correctness property given as a temporal-logic formula. Model checking has enjoyed considerable success in application to hardware designs. Progress is now being seen in other application domains such as software and protocol verification.

## 2. Classical roots

Process algebra can be viewed as a generalization of the classical theory of formal languages and automata [4], focusing on system specification and behavior rather than language recognition and generation. Process algebra also embodies the principles of cellular automata [5] – cells receiving inputs from neighboring cells and then taking appropriate action – while adding a notion of programmability: nondeterminism, dynamic topologies, evolving cell behavior, etc.

Process algebra lays the groundwork for a rigorous system-design ideology, providing support for specification, verification, implementation, testing and other life-cycle-critical activities. Interest in process algebra, however, extends beyond the system-design arena, to areas such as programming language design and semantics, complexity theory, real-time programming, and performance modeling and analysis.

## 3. About this Handbook

This Handbook documents the fate of process algebra from its modern inception in the late 1970's to the present. It is intended to serve as a reference source for researchers, students, and system designers and engineers interested in either the theory of process algebra or in learning what process algebra brings to the table as a formal system description and verification technique.

The Handbook is divided into six parts, the first five of which cover various theoretical and foundational aspects of process algebra. Part 6, the final part, is devoted to tools for applying process algebra and to some of the applications themselves. Each part contains between two and four chapters. Chapters are self-contained and can be read independently of each other. In total, there are 19 chapters spanning roughly 1300 pages. Collectively, the Handbook chapters give a comprehensive, albeit necessarily incomplete, view of the field.

Part 1, consisting of four chapters, covers a broad swath of the **basic theory** of process algebra. In Chapter 1, *The Linear Time – Branching Time Spectrum I*, van Glabbeek gives

a useful structure to, and an encyclopedic account of, the many behavioral relations that have been proposed in the process-algebra literature. Chapter 2, *Trace-Oriented Models of Concurrency* by Broy and Olderog, provides an in-depth presentation of trace-oriented models of process behavior, where a trace is a communication sequence that a process can perform with its environment. Aceto, Fokkink and Verhoef present a thorough account of *Structural Operational Semantics* in Chapter 3. Part 1 concludes with Chapter 4, *Modal Logics and Mu-Calculi: An Introduction* by Bradfield and Stirling. Modal logics, which extend classical logic with operators for possibility and necessity, play an important role in filling out the semantic picture of process algebra.

Part 2 is devoted to the sub-specialization of process algebra known as **finite-state processes**. This class of processes holds a strong practical appeal as finite-state systems can be verified in an automatic, push-button style. The two chapters in Part 2 address finite-state processes from an axiomatic perspective: Chapter 5, *Process Algebra with Recursive Operations* by Bergstra, Fokkink and Ponse; and from an algorithmic one: Chapter 6, *Equivalence and Preorder Checking for Finite-State Systems* by Cleaveland and Sokolsky.

**Infinite-state processes**, the subject of Part 3, capture process algebra at its most expressive. Chapter 7, the first of the three chapters in this part, *A Symbolic Approach to Value-Passing Processes* by Ingólfsdóttir and Lin, systematically examines the class of infinite-state processes arising from the ability to transmit data from an arbitrary domain of values. Symbolic techniques are proposed as a method for analyzing such systems. Chapter 8, by Parrow, is titled *An Introduction to the $\pi$-Calculus*. This chapter investigates the area of mobile processes, an enriched form of value-passing process that is capable of transmitting communication channels and even processes themselves from one process to another. Finally, Burkhart, Caucal, Moller and Steffen consider the equivalence-checking and model-checking problems for a large variety of infinite-state processes in Chapter 9, *Verification on Infinite Structures*.

The three chapters of Part 4 explore several **extensions to process algebra** that make it easier to model the kinds of systems that arise in practice. Chapter 10 focuses on real-time systems. *Process Algebra with Timing: Real Time and Discrete Time* by Middelburg and Baeten, presents a real-time extension of the process algebra ACP that extends ACP in a natural way. The final two chapters of Part 4 study the impact on process algebra of replacing the standard notion of "nondeterministically choose the next transition to execute" with one in which probability or priority information play pivotal roles. Chapter 11, *Probabilistic Extensions of Process Algebras* by Jonsson, Larsen and Yi, targets the probabilistic case, which is especially useful for modeling system failure, reliability, and performance. Chapter 12, *Priority in Process Algebra* by Cleaveland, Lüttgen and Natarajan, considers the case of priority, and shows how a process algebra with priority can be used to model interrupts, prioritized choice and real-time behavior.

Process algebra was originally conceived with the view that concurrency equals interleaving. That is, the concurrent execution of a collection of events can be modeled as their interleaved execution, in any order. More recent versions of process algebra known as **non-interleaving process algebras**, aim to model concurrency directly, for example, as embodied in Petri nets. The four chapters of Part 5 address this subject. Chapter 13, *Partial-Order Process Algebra* by Baeten and Basten, thoroughly considers the impact of a non-interleaving semantics on ACP. Chapter 14, *A Unified Model for Nets and Process*

*Algebras* by Best, Devillers and Koutny, examines a range of issues that arise when process algebra and Petri nets are combined together. Another kind of non-interleaving treatment of concurrency is put forth in Chapter 15, Castellani's *Process Algebras with Localities*. In this approach, "locations" are assigned to parallel components, resulting in what Castellani calls a "distributed semantics" for process algebra. Finally, in Chapter 16, Gorrieri and Rensink's *Action Refinement* gives a thorough treatment of process algebra with action refinement, the operation of replacing a high-level atomic action with a low-level process. The interplay between action refinement and non-interleaving semantics is carefully considered.

Part 6, the final part of the Handbook, contains three chapters dealing with **tools and applications** of process algebra. The first of these, Chapter 17, *Algebraic Process Verification* by Groote and Reniers, gives a close-up account of verification techniques for distributed algorithms and protocols, using process algebra extended with data ($\mu$CRL). Chapter 18, *Discrete Time Process Algebra and the Semantics of SDL* by Bergstra, Middelburg and Usenko, introduces a discrete-time process algebra that is used to provide a formal semantics for SDL, a widely used formal description technique for telecommunications protocols. Finally, Chapter 19, *A Process Algebra for Interworkings* by Mauw and Reniers, devises a process-algebra-based semantics for Interworkings, a graphical design language of Philips Kommunikations Industrie.

## Acknowledgements

The editors gratefully acknowledge the constant support of Arjen Sevenster, our manager at Elsevier; without his efforts, this Handbook would not have seen the light of day. We are equally grateful to all the authors; their diligence, talent, and patience are greatly appreciated. We would also like to thank the referees, whose reports significantly enhanced the final contents of the Handbook. They are: Luca Aceto, Jos Baeten, Wan Fokkink, Rob Goldblatt, Hardi Hungar, Joost-Pieter Katoen, Alexander Letichevsky, Bas Luttik, Faron Moller, Uwe Nestmann, Nikolaj Nikitchenko, Benjamin Pierce, Piet Rodenburg, Mariëlle Stoelinga, P.S. Thiagarajan, and Yaroslav Usenko. Finally, we would like to thank Rance Cleaveland for his help in writing this preface.

Autumn 2000
Jan A. Bergstra (Amsterdam),
Alban Ponse (Amsterdam),
Scott A. Smolka (Stony Brook, New York)

## References

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

[2] S.D. Brookes, C.A.R. Hoare and A.W. Roscoe, *A theory of communicating sequential processes*, J. ACM **31** (3) (1984), 560–599.

[3] E.M. Clarke, E.A. Emerson and A.P. Sistla, *Automatic verification of finite-state concurrent systems using temporal logic specifications*, ACM TOPLAS **8** (2) (1986).

[4] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley (1979).

[5] J. von Neumann, *Theory of self-reproducing automata*, A.W. Burks, ed., Urbana, University of Illinois Press (1966).

[6] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Comput. Sci. 92, Springer-Verlag (1980).

[7] G.D. Plotkin, *A structural approach to operational semantics*, Report DAIMI FN-19, Computer Science Department, Aarhus University (1981).

Jan A. Bergstra[2,3], Alban Ponse[1,2], Scott A. Smolka[4]

[1] *CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands*
*http://www.cwi.nl/*

[2] *University of Amsterdam, Programming Research Group, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands*
*http://www.science.uva.nl/research/prog/*

[3] *Utrecht University, Department of Philosophy, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands*
*http://www.phil.uu.nl/eng/home.htmlE-mail:*

[4] *State University of New York at Stony Brook, Department of Computer Science*
*Stony Brook, NY 11794-4400, USA*
*http://www.cs.sunysb.edu/*

*E-mails: janb@science.uva.nl, alban@science.uva.nl, sas@cs.sunysb.edu*