Eike Best
Raymond Devillers
Maciej Koutny

Petri Net Algebra

Springer

Eike Best
Raymond Devillers
Maciej Koutny

Petri Net Algebra

With 111 Figures

Springer

Authors

Prof. Dr. Eike Best
Fachbereich Informatik
Carl von Ossietzky Universität
Oldenburg
26111 Oldenburg, Germany
Eike.Best@informatik.uni-oldenburg.de

Prof. Dr. Raymond Devillers
Faculté des Sciences
Laboratoire d'Informatique Théorique
Université Libre de Bruxelles
1050 Bruxelles, Belgium
rdevil@ulb.ac.be

Prof. Dr. Maciej Koutny
Department of Computing Science
University of Newcastle
Newcastle upon Tyne NE1 7RU, U.K.
Maciej.Koutny@newcastle.ac.uk

Series Editors

Prof. Dr. Wilfried Brauer
Institut für Informatik,
Technische Universität München
Arcisstraße 21, 80333 München
Germany
Brauer@informatik.tu-muenchen.de

Prof. Dr. Grzegorz Rozenberg
Leiden Institute
of Advanced Computer Science
University of Leiden
Niels Bohrweg 1, 2333 CA Leiden
The Netherlands
rozenber@liacs.nl

Prof. Dr. Arto Salomaa
Turku Centre for Computer Science
Lemminkäisenkatu 14 A, 20520 Turku
Finland
asalomaa@utu.fi

Library of Congress Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Best, Eike:
Petri net algebra / Eike Best ; Raymond Devillers ; Maciej Koutny. -
Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ;
Milan ; Paris ; Singapore ; Tokyo : Springer, 2001
(Monographs on theoretical computer science)
ISBN 3-540-67398-9

ACM Computing Classification (1998): F.3.2, F.1.1–2, D.2.2, G.2

ISBN 3-540-67398-9 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York,
a member of BertelsmannSpringer Science+Business Media GmbH

Printed in Germany

Production: PRO EDIT GmbH, Heidelberg
Cover Design: *design & production* GmbH, Heidelberg
Typesetting: Camera ready by authors
Printed on acid-free paper SPIN: 10765212 45/3142/hs – 5 4 3 2 1 0

Monographs in Theoretical Computer Science
An EATCS Series

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Preface

In modern society services and support provided by computer-based systems have become ubiquitous and indeed have started to fundamentally alter the way people conduct their business. Moreover, it has become apparent that among the great variety of computer technologies available to potential users a crucial role will be played by concurrent systems. The reason is that many commonly occurring phenomena and computer applications are highly concurrent: typical examples include control systems, computer networks, digital hardware, business computing, and multimedia systems. Such systems are characterised by ever increasing complexity, which results when large numbers of concurrently active components interact. This has been recognised and addressed within the computing science community. In particular, several formal models of concurrent systems have been proposed, studied, and applied in practice.

This book brings together two of the most widely used formalisms for describing and analysing concurrent systems: Petri nets and process algebras. On the one hand, process algebras allow one to specify and reason about the design of complex concurrent computing systems by means of algebraic operators corresponding to common programming constructs. Petri nets, on the other hand, provide a graphical representation of such systems and an additional means of verifying their correctness efficiently, as well as a way of expressing properties related to causality and concurrency in system behaviour. The treatment of the structure and semantics of concurrent systems provided by these two types of models is different, and it is thus virtually impossible to take full advantage of their overall strengths when they are used separately.

The idea of combining Petri nets and process algebras can be traced back to the early 1970s. More directly, this book builds on work carried out by its authors within two EU-funded research projects. It presents a step-by-step development of a rigorous framework for the specification and verification of concurrent systems, in which Petri nets are treated as composable objects, and as such are embedded in a general process algebra. Such an algebra is given an automatic Petri net semantics so that net-based verification techniques, based on structural invariants and causal partial orders, can be applied. The book contains full proofs and carefully chosen examples, and

describes several possible directions for further research. A unique aspect is that the development of the Petri net algebra is handled so as to allow for further application-oriented extensions and modifications.

The book is primarily aimed at researchers, lecturers, and graduate students interested in studying and applying formal methods for concurrent computing systems. It is self-contained in the sense that no previous knowledge of Petri nets and process algebras is required, although we assume that the reader is familiar with basic concepts and notions of set theory and graph theory.

We would like to express our deepest gratitude to our friends and colleagues with whom we conducted the work presented in this monograph. In particular, we would like to thank Javier Esparza, Hans Fleischhack, Bernd Grahlmann, Jon G. Hall, Richard P. Hopkins, Hanna Klaudel, and Elisabeth Pelz. We would also like to thank Wilfried Brauer for his support and encouragement during the writing of this book, Hans Wössner for his excellent editorial advice, and an anonymous reviewer for several very useful comments and suggestions. Last but not least, we would like to thank our respective families for their unfailing support.

December 2000,

Oldenburg — Eike Best
Bruxelles — Raymond Devillers
Newcastle upon Tyne — Maciej Koutny

This book was typeset with Leslie Lamport's LaTeX document preparation system, which is itself based on Donald Knuth's TeX. To produce diagrams, we used the *graphs* macro package by Frank Drewes.

Contents

1. Introduction 1

2. The Petri Box Calculus 7
- 2.1 An Informal Introduction to CCS 8
- 2.2 An Informal Introduction to Petri Nets 9
- 2.3 The Structure and Behaviour of PBC Expressions 11
- 2.4 Sequential Composition 14
- 2.5 Synchronisation 15
- 2.6 Synchronisation and Parallel Composition 21
- 2.7 Other PBC Operators 24
- 2.8 Modelling a Concurrent Programming Language 25
- 2.9 Literature and Background 28

3. Syntax and Operational Semantics 29
- 3.1 Standard PBC Syntax 29
- 3.2 Structured Operational Semantics 33
  - 3.2.1 The Basic Setup 35
  - 3.2.2 Equivalence Notions 37
  - 3.2.3 Elementary Actions 41
  - 3.2.4 Parallel Composition 42
  - 3.2.5 Choice Composition 45
  - 3.2.6 Sequential Composition 46
  - 3.2.7 Synchronisation 47
  - 3.2.8 Standard PBC Synchronisation 47
  - 3.2.9 Auto-synchronisation and Multilink-synchronisation 50
  - 3.2.10 Step-synchronisation 52
  - 3.2.11 Basic Relabelling 53
  - 3.2.12 Restriction 54
  - 3.2.13 Scoping 55
  - 3.2.14 Iteration 58
  - 3.2.15 Recursion 59
- 3.3 Extensions 62
  - 3.3.1 Generalised Iterations 62
  - 3.3.2 Data Variables 64

3.3.3 Generalised Control Flow Operators 66
3.3.4 Generalised Communication Interface Operators 67
3.4 Extended PBC Syntax 69
3.5 Examples of Transition Systems 69
3.6 Literature and Background 71

4. Petri Net Semantics 73
4.1 Compositionality and Nets 73
4.2 Labelled Nets and Boxes 75
4.2.1 An Example 75
4.2.2 Actions and Relabellings 76
4.2.3 Labelled Nets 77
4.2.4 Equivalence Notions 82
4.2.5 Boxes 86
4.3 Net Refinement 90
4.3.1 Operator Boxes 90
4.3.2 Intuition Behind Net Refinement 92
4.3.3 Place and Transition Names 95
4.3.4 Formal Definition of Net Refinement 97
4.3.5 Remarks on Net Refinement 99
4.3.6 Properties 101
4.3.7 Discussion 103
4.4 Petri Net Semantics of PBC 104
4.4.1 Elementary Actions 106
4.4.2 Parallel Composition 106
4.4.3 Choice Composition 107
4.4.4 Sequential Composition 109
4.4.5 Basic Relabelling 110
4.4.6 Synchronisation 110
4.4.7 Restriction 119
4.4.8 Scoping 120
4.4.9 Iteration 120
4.4.10 Data Variables 123
4.4.11 Generalised Control Flow Operators 124
4.4.12 Generalised Communication Interface Operators 126
4.4.13 Generalised Iterations 127
4.5 Refined Operators 129
4.6 Literature and Background 132

5. Adding Recursion 133
5.1 Inclusion Order on Labelled Nets 133
5.2 Solving Recursive Equations 135
5.2.1 Using Fixpoints to Solve Recursive Equations 137
5.2.2 Places and Transitions in Net Solutions 140
5.2.3 An Example of the Limit Construction 144

5.2.4 Deriving Seed Boxes ... 145
5.2.5 A Closed Form of the Maximal Solution ... 151
5.2.6 Minimal Solutions ... 153
5.3 Finitary Equations and Finite Operator Boxes ... 157
5.3.1 Finitary Equation ... 157
5.3.2 Finite Operator Box ... 159
5.4 Further Examples ... 161
5.4.1 Unbounded Parallel Composition ... 161
5.4.2 Rear-unguardedness ... 162
5.4.3 Concurrency Within Unbounded Choice ... 164
5.4.4 Extreme Unguardedness ... 166
5.4.5 (Non)use of Empty Nets in the Limit Construction ... 167
5.5 Solving Systems of Recursive Equations ... 167
5.5.1 Approximations, Existence, and Uniqueness ... 168
5.5.2 A Closed Form of the Maximal Solution ... 169
5.5.3 Guarded Systems ... 171
5.6 Literature and Background ... 172

6. S-invariants ... 173
6.1 S-invariants, S-components, and S-aggregates ... 174
6.1.1 S-invariants ... 176
6.1.2 S-components ... 181
6.1.3 S-aggregates ... 182
6.2 The Synthesis Problem for Net Refinement ... 183
6.2.1 Composing S-invariants ... 185
6.2.2 Multiplicative Distribution Functions ... 190
6.2.3 Ex-binary S-invariants ... 193
6.2.4 Rational Groupings ... 195
6.3 The Synthesis Problem for Recursive Systems ... 200
6.3.1 Name Trees of Nets in the Maximal Solution ... 201
6.3.2 Composing S-invariants for Recursive Boxes ... 202
6.3.3 Coverability Results ... 209
6.4 Finite Precedence Properties ... 216
6.4.1 Process Semantics ... 218
6.4.2 Finite Precedence of Events ... 222
6.4.3 Finiteness of Complete Processes ... 225
6.5 Literature and Background ... 226

7. The Box Algebra ... 227
7.1 SOS-operator Boxes ... 227
7.1.1 A Running Example ... 232
7.1.2 Properties of Factorisations ... 232
7.1.3 The Domain of Application of an SOS-operator Box ... 234
7.1.4 Static Properties of Refinements ... 235
7.1.5 Markings of Nets ... 239

7.2 Structured Operational Semantics of Composite Boxes . . . 241
7.2.1 Soundness . . . 243
7.2.2 Similarity Relation on Tuples of Boxes . . . 245
7.2.3 Completeness . . . 248
7.2.4 Solutions of Recursive Systems . . . 253
7.2.5 Behavioural Restrictions . . . 255
7.3 A Process Algebra and its Semantics . . . 259
7.3.1 A Running Example: the DIY Algebra . . . 262
7.3.2 Infinite Operators . . . 264
7.3.3 Denotational Semantics . . . 267
7.3.4 Structural Similarity Relation on Expressions . . . 270
7.3.5 Transition-based Operational Semantics . . . 279
7.3.6 Consistency of the Two Semantics . . . 286
7.3.7 Label-based Operational Semantics . . . 287
7.3.8 Partial Order Semantics of Box Expressions . . . 290
7.4 Literature and Background . . . 294

8. PBC and Other Process Algebras . . . 295
8.1 (Generalised) PBC is a Box Algebra . . . 295
8.1.1 PBC Without Loops . . . 295
8.1.2 Safe Translation of the Ternary PBC Iteration . . . 299
8.1.3 PBC with Generalised Loops . . . 306
8.2 Other Process Algebras . . . 308
8.2.1 CCS . . . 310
8.2.2 TCSP . . . 311
8.2.3 COSY . . . 311
8.3 Literature and Background . . . 312

9. A Concurrent Programming Language . . . 313
9.1 Syntax of *Razor* . . . 313
9.1.1 Programs and Blocks . . . 315
9.1.2 Declarations . . . 315
9.1.3 Commands and Actions . . . 316
9.1.4 Guarded Commands . . . 316
9.1.5 Expressions and Operators . . . 317
9.1.6 Syntactic Variations . . . 317
9.2 Semantics of *Razor* . . . 318
9.2.1 Programs, Blocks, and Declarations . . . 319
9.2.2 Basic Channel Processes . . . 321
9.2.3 Command Connectives . . . 324
9.2.4 Actions and Guarded Commands . . . 325
9.3 Three *Razor* Programs . . . 329
9.4 Adding Recursive Procedures . . . 332
9.5 Some Consequences of the Theory . . . 336

9.6 Proofs of Distributed Algorithms 340
9.6.1 A Final Set of Petri-Net-Related Definitions 340
9.6.2 Peterson's Mutual Exclusion Algorithm 342
9.6.3 Dekker's and Morris's Mutual Exclusion Algorithms .. 346
9.7 Literature and Background 347

10. Conclusion 349

Appendix: Solutions of Selected Exercises 351

References 362

Index 369

1. Introduction

Computing Science is about building systems. Many of such systems, and an increasing number of them, are concurrent or distributed. That may be so, for instance, because more than one processors inhabit a single machine, or because several machines at different locations are made to cooperate with each other, or because dedicated processing elements are built into a single system.

Concurrency theory is sufficiently wide and difficult a subject of Computing Science such that, to this date, no single mathematical formalisation of it, of which there exist many, can claim to have absolute priority over others. Much research has gone into exploring individual concurrency theoretical approaches and exploiting their particular advantages. The state of the art is such that many formalisms have been developed to the extent of being very useful, and actually used, in practice. Other research has gone into attempting to combine the relative advantages of two or more, originally perhaps competing, approaches. Such research, it may be hoped, would lay foundations such that in some not too distant day, a uniform and useful generalised formalism emerges that comprises the most important advantages of the present ones.

This book is about combining two widely known and well studied theories of concurrency: *process algebras* and *Petri nets*. Advantages of process algebras include the following ones:

- They allow the study of connectives directly related to actual programming languages.
- They are compositional by definition. Compositionality refers to the ability of composing larger systems from smaller ones in a structured way.
- They come with a variety of concomitant and derived logics. Such logics may facilitate reasoning about important properties of systems.
- They come with a variety of algebraic laws. Such laws may be used to manipulate systems; in particular, to refine them, or to prove them correct with respect to some specification.

Advantages of Petri nets are the following, amongst others:

- Petri nets sharply distinguish between states and activities (the latter being defined as changes of state). This corresponds to the distinction between places (local states) and transitions (local activities).
- Global states and global activities are not basic notions, but are derived from their local counterparts. This is very intuitive and suits well the description of a distributed system as the sum of its local parts. It also facilitates the description of concurrent behaviour by partially ordered sets.
- While being satisfactorily formal, Petri nets also come with a graphical representation which is easy to grasp and has therefore a wide appeal for practitioners interested in applications.
- By their representation as bipartite graphs, Petri nets have useful links both to graph theory and to linear algebra. Both links can be exploited for the verification of systems.

Trying to achieve a full combination of all the advantages just mentioned would certainly be an ambitious task. This book's aim is more modest in that it only attempts to forge links between a fundamental (but restricted) class of Petri nets and a basic (but again restricted) process algebra. However, the book does attempt to make these links as tight and strong as possible. The core plan of this book is to investigate the structural and behavioural aspects of these two basic models, and its main achievement, perhaps, is the revelation that there is, in fact, an extremely strong equivalence between them.

The basic process algebraic setup we take as our starting point is the Petri Box Calculus (PBC [7]), which in itself is a variant of Robin Milner's Calculus of Communicating Systems (CCS [80, 81]). There are many other incarnations of process algebra [2, 54, 59, 62], some of which will be described in due course in this book. The reason why we prefer CCS as a starting point is that, of all the mentioned ones, it seems to be most directly useful for describing the meaning of a concurrent programming language, which is that by which most concurrent systems are actually implemented.

The basic Petri net theoretic setup we take as our starting point is the place/transition net model [90]. Again, this is but one of the many net-based models that have been investigated [15, 63]. However, it is a fundamental one of which the others are extensions or variations. Also, place/transition nets share with CCS-like process algebras the property of being rather straightforwardly useful for concurrent programming languages [3] (and, more generally, for concurrent algorithms [92]).

The idea to combine a process algebra such as CCS with a Petri net model such as place/transition nets is not new. Perhaps the first author advocating such an approach was Peter Lauer in [71], attributing it there to Brian Randell. In the years between this early memorandum and now, many articles and theses [45], and also several books, have been written with exactly this connection in mind. In [95], Dirk Taubner gives high-level Petri net semantics

to a process algebra which is akin to CCS. In [84], Ernst-Rüdiger Olderog uses (place/transition) Petri net semantics of another process algebraic variant as a stepping stone in the design of concurrent systems. In [62], place/transition nets are used as semantics of yet another process algebra.

All these setups subtly differ from each other, and they are also limited in their various individual ways. For instance, recursion is absent in [62], and is treated nondenotationally and limited to guarded cases in [84, 95]. More importantly, perhaps, than these differences is the fact that none of the cited works has demonstrated a very strong equivalence ranging from the set of operators in each framework to the generated behaviour, and holding in full generality.[1] It is the desire for such an equivalence which distinguishes the present work from the other ones mentioned above, and we achieve it at the expense of neglecting other valid aims, such as extensions to high-level nets or to other process-algebraic setups, and also the treatment of substantial case studies.

In the course of searching for the desired strong connections, the authors have experienced several surprises. One germ to such a surprise was the separation, already proposed in PBC, of the CCS synchronisation operator into parallel composition and (the communication part of the) synchronisation. This has eventually led to a general classification of all operators into place-based ones and transition-based ones; note that there needs to be a close connection between process algebras and Petri nets, just in order to formulate such a classification.

An advantage of this classification is that all operators within each class can be treated similarly. An exception to this rule is recursion, which can be seen as belonging to the first class, but needs a more involved treatment than other place-based operators. But, ultimately, all these operators are based on a single net-theoretic operation: transition refinement. (It may seem odd to the reader that place-based operators should be based on a transition-related operation, but the simple explanation is that the environment of a refined transition, which is a set of places, plays an important role in such a definition.) Likewise, all operators in the second class, i.e., transition-based ones, are based on a single operation: relabelling. This is not actually an indigenous Petri net operation; rather, it refers to the transition labelling of a net. It can be defined in a sufficiently powerful way so that all operators of the second class become special instances of relabelling. As a matter of fact, in a formal treatment, we will integrate refinement and relabelling into only one operation, of which everything else is just a particular case.

[1] In one of them, it has in fact been conjectured (see [84], conjecture 3.7.6) that it is impossible to achieve such an equivalence. This conjecture, along with the fact that in our setup we do achieve such an equivalence, shows what far-reaching consequences such subtle and innocent-looking distinctions may have, and what scope there is for fine tuning and adjusting existing theories until, perhaps, an optimal one eventually emerges.

Viewing all operators as incarnations of such a powerful operation leads to a further generalisation. This is so because refining a transition in a net can be viewed as a function from nets to nets. For instance, sequential composition can be viewed as a very simple two-transition net which takes two nets as arguments and creates another one (their sequence) as a result. Nets used in this way as functions from (tuples of) nets to nets will be called Operator Boxes. Usually, in a process algebra, a handful of operators (each corresponding to a particular operator box) is considered. However, because of the generalised setup described above, we may now formulate all our results in terms of a very large class of operator boxes, all of which could be interpreted as process-algebraic operators. It is one of the principal results of this book to state precisely which of such nets can, and which cannot, be viewed as good operators.

The main criterion for this, in fact, will be whether or not the desired (behavioural) equivalence does hold. What we desire is that the entire reachability graph, that is, the set of reachable states, together with the information as to which of them are reachable from others, is exactly the same, whether one formalises it in terms of process algebra or in terms of nets. In order to achieve this equivalence, we will employ a net-independent behavioural semantics of process algebra (so-called SOS semantics, for structured operational semantics [89]) and compare it with the net-induced semantics. The equivalence of both, for a large class of operator boxes, is the central result in Chap. 7 of this book.

The structure of the book is as follows:

Chapter 2 is introductory, focussing, in particular, on the above-mentioned separation between synchronisation and concurrent composition. It also serves to introduce CCS, Petri nets, PBC, and its SOS semantics. Moreover, the penultimate section of Chap. 2 contains an intuitive motivation why CCS and PBC are used for concurrent programming languages. The entire chapter is written in a fairly informal style to make for easy reading. In passing, Chap. 2 introduces some of the basic notions used throughout the book.

In Chap. 3, the behavioural SOS semantics of PBC is defined. The style of giving this semantics is adapted to concurrent evolutions, already anticipating the desired equivalence result.

In Chap. 4, a Petri net semantics of PBC is developed. This semantics is based on a special kind of labelled place/transition nets, which makes them amenable to being composed. Instead of defining separately, for each operator, the rules to follow in order to derive nets corresponding to expressions constructed from it, a general simultaneous refinement and relabelling operation is defined together with a notion of operator box.

In Chap. 5, the treatment of recursive expressions is dealt with using a general fixpoint theory developed in order to solve recursive equations on Petri nets.

In Chap. 6, a general analysis of nets obtained through refinements or recursive equations is conducted, based on the structural notion of S-invariants. Many interesting results are obtained in this way concerning their behaviour (such as safeness, or lack of auto-concurrency), together with the facts that complete evolutions are always finite, and that extending the Petri net semantics to infinite steps would not lead to a substantially increased expressive power.

In Chap. 7, the SOS and Petri net semantics of Chaps. 3, 4, and 5 are taken up more directly. In particular, the equivalence of these two approaches to define the semantics of our algebra is proved formally. Moreover, the proof is done in such a general and generic setting that it will not be necessary to re-evaluate the whole theory if, at a later stage, it is desired to introduce new operators or basic processes, provided some straightforward conditions are fulfilled.

In Chap. 8, an argument is made that PBC, and indeed other process algebras, are instances of the general framework developed in the previous chapter.

In Chap. 9, the applicability of the theory developed so far is finally demonstrated by giving a compositional semantics for a concurrent programming language. The emphasis is not so much on the novelty of this approach, but on the ease with which the translations can be given, and the results proved. The hope is expressed (and some examples are given that may be seen as supporting this view) that thanks to such an ease, it will be possible to use process-algebraic and net-theoretic methods harmoniously side-by-side in the verification of important properties of algorithms expressed by concurrent programs.

Chapter 10 summarises the material covered by this book and looks at the possible future directions of research, while the Appendix contains solutions of some selected exercises. Besides references, the book also contains an index which enables the reader to find definitions and places of use of important terms and notations.

This work builds on previous work conducted by two European research projects, DEMON (Esprit Basic Research Action 3148) and CALIBAN (Esprit Basic Research Working Group 6067), see [7, 6, 10, 11, 12, 8, 32, 25, 30, 31, 36, 40, 68, 69, 70]; and, later, within the ARC Project 1032 BAT (Box Algebra with Time). The authors gratefully acknowledge contributions which are due to many discussions with numerous researchers, in particular with Javier Esparza, Hans Fleischhack, Bernd Grahlmann, Jon G. Hall, Richard P. Hopkins, Hanna Klaudel, and Elisabeth Pelz.