The $\pi$-calculus: a Theory of Mobile Processes

Davide Sangiorgi

INRIA Sophia Antipolis

David Walker

University of Oxford
Quelli che s’innamoran di pratica sanza scientia
son come ’l nocchiere ch’entra in navilio sanza timone o bussola,
che mai ha certezza dove si vada.

– Leonardo da Vinci
Contents

Foreword ix
Preface xi
General Introduction 1

Part I: The \( \pi \)-calculus 5
Introduction to Part I 7

1 Processes .................................................. 11
1.1 Syntax .................................................. 11
1.2 Reduction ............................................. 17
1.3 Action .................................................. 36
1.4 Basic properties of the transition system .............. 44

2 Behavioural Equivalence ................................. 54
2.1 Strong barbed congruence ............................. 54
2.2 Strong bisimilarity .................................... 64
2.3 Up-to techniques ..................................... 80
2.4 Barbed congruence ................................... 92

Notes and References for Part I 118

Part II: Variations of the \( \pi \)-calculus 121
Introduction to Part II 123

3 Polyadicity and Recursion ............................. 127
3.1 Polyadicity ............................................ 127
3.2 Recursion ............................................. 132
3.3 Priority-queue data structures ........................................... 138
3.4 Data as processes .......................................................... 145

4 Behavioural Equivalence, continued ................................. 154
4.1 Distinctions ................................................................. 154
4.2 Variants of bisimilarity .................................................... 157
4.3 The late transition relations ............................................. 158
4.4 Ground bisimilarity ....................................................... 162
4.5 Late bisimilarity ............................................................ 164
4.6 Open bisimilarity .......................................................... 166
4.7 The weak equivalences .................................................... 172
4.8 Axiomatizations and proof systems .................................. 174

5 Subcalculi ........................................................................ 189
5.1 The Asynchronous \( \pi \)-calculus ...................................... 189
5.2 Syntax of A\( \pi \) ............................................................ 190
5.3 Behavioural equivalence in A\( \pi \) ............................... 194
5.4 Asynchronous equivalences .............................................. 198
5.5 Expressiveness of asynchronous calculi ......................... 203
5.6 The Localized \( \pi \)-calculus ............................................ 211
5.7 Internal mobility ............................................................ 215
5.8 Non-congruence results for ground bisimilarity ............... 223

Notes and References for Part II ........................................ 227

Part III: Typed \( \pi \)-calculi ...................................................... 231

Introduction to Part III ....................................................... 233

6 Foundations ....................................................................... 236
6.1 Terminology and notation for typed calculi .................... 236
6.2 Base-\( \pi \) ................................................................. 238
6.3 Properties of typing ....................................................... 244
6.4 The simply-typed \( \pi \)-calculus ...................................... 247
6.5 Products, unions, records, and variants ......................... 249
6.6 Pattern matching in input .............................................. 255
6.7 Recursive types .............................................................. 257

7 Subtyping ........................................................................... 260
7.1 i/o types ................................................................. 261
7.2 Properties of the type systems with i/o ........................... 265
7.3 Other subtyping ............................................................. 270
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4</td>
<td>The priority queues, revisited</td>
<td>272</td>
</tr>
<tr>
<td>7.5</td>
<td>Encodings between union and product types</td>
<td>276</td>
</tr>
<tr>
<td>8</td>
<td>Advanced Type Systems</td>
<td>281</td>
</tr>
<tr>
<td>8.1</td>
<td>Linearity</td>
<td>281</td>
</tr>
<tr>
<td>8.2</td>
<td>Receptiveness</td>
<td>288</td>
</tr>
<tr>
<td>8.3</td>
<td>Polymorphism</td>
<td>296</td>
</tr>
<tr>
<td></td>
<td>Notes and References for Part III</td>
<td>305</td>
</tr>
</tbody>
</table>

### Part IV: Reasoning about Processes using Types

#### Introduction to Part IV

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Groundwork</td>
<td>311</td>
</tr>
<tr>
<td>9.1</td>
<td>Using types to obtain encapsulation</td>
<td>313</td>
</tr>
<tr>
<td>9.2</td>
<td>Why types for reasoning?</td>
<td>316</td>
</tr>
<tr>
<td>9.3</td>
<td>A security property</td>
<td>317</td>
</tr>
<tr>
<td>9.4</td>
<td>Typed behavioural equivalences</td>
<td>319</td>
</tr>
<tr>
<td>9.5</td>
<td>Equivalences and preorders in simply-typed $\pi$-calculus</td>
<td>328</td>
</tr>
<tr>
<td>10</td>
<td>Behavioural Effects of i/o Types</td>
<td>329</td>
</tr>
<tr>
<td>10.1</td>
<td>Type coercion</td>
<td>329</td>
</tr>
<tr>
<td>10.2</td>
<td>Examples</td>
<td>330</td>
</tr>
<tr>
<td>10.3</td>
<td>Wires in the Asynchronous $\pi$-calculus</td>
<td>335</td>
</tr>
<tr>
<td>10.4</td>
<td>Delayed input</td>
<td>336</td>
</tr>
<tr>
<td>10.5</td>
<td>Sharpened Replication Theorems</td>
<td>338</td>
</tr>
<tr>
<td>10.6</td>
<td>Proof techniques</td>
<td>340</td>
</tr>
<tr>
<td>10.7</td>
<td>Context Lemma</td>
<td>342</td>
</tr>
<tr>
<td>10.8</td>
<td>Adding internal mobility</td>
<td>348</td>
</tr>
<tr>
<td>11</td>
<td>Techniques for Advanced Type Systems</td>
<td>351</td>
</tr>
<tr>
<td>11.1</td>
<td>Some properties of linearity</td>
<td>351</td>
</tr>
<tr>
<td>11.2</td>
<td>Behavioural properties of receptiveness</td>
<td>352</td>
</tr>
<tr>
<td>11.3</td>
<td>A proof technique for polymorphic types</td>
<td>359</td>
</tr>
</tbody>
</table>

#### Notes and References for Part IV

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Notes and References for Part III</td>
<td>365</td>
</tr>
</tbody>
</table>

### Part V: The Higher-Order Paradigm

#### Introduction to Part V

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Higher-Order $\pi$-calculus</td>
<td>373</td>
</tr>
<tr>
<td>12.1</td>
<td>Simply-typed HO$\pi$</td>
<td>373</td>
</tr>
</tbody>
</table>
Contents

12.2 Other HO\pi languages .............................................. 381

13 Comparing First-Order and Higher-Order Calculi .............. 383
13.1 Compiling higher order into first order ....................... 383
13.2 Optimizations .................................................. 397
13.3 Reversing the compilation ....................................... 408
13.4 Full abstraction ................................................ 412

Notes and References for Part V ....................................... 415

Part VI: Functions as Processes ............................................. 419

Introduction to Part VI .................................................. 421

14 The \lambda-calculus ................................................. 424
14.1 The formal system ................................................. 424
14.2 Contrasting \lambda and \pi ......................................... 426
14.3 Reduction strategies: call-by-name, call-by-value, call-by-need....... 429

15 Interpreting \lambda-calculi ............................................ 434
15.1 Continuation Passing Style ........................................ 434
15.2 Notations and terminology for functions as processes .............. 436
15.3 The interpretation of call-by-value ................................ 438
15.4 The interpretation of call-by-name ................................ 452
15.5 A uniform encoding .............................................. 461
15.6 Optimizations of the call-by-name encoding ...................... 464
15.7 The interpretation of strong call-by-name ....................... 465

16 Interpreting Typed \lambda-calculi .................................... 469
16.1 Typed \lambda-calculus ............................................ 469
16.2 The interpretation of typed call-by-value ....................... 470
16.3 The interpretation of typed call-by-name ....................... 474

17 Full Abstraction .................................................... 477
17.1 The full-abstraction problem ...................................... 477
17.2 Applicative bisimilarity ........................................ 478
17.3 Soundness and non-completeness ................................ 479
17.4 Extending the \lambda-calculus ..................................... 483

18 The Local Structure of the Interpretations ...................... 492
18.1 Sensible theories and lazy theories ............................. 492
18.2 \Le{\nu}vy-Longo Trees ........................................... 493
18.3 The Local Structure Theorem for call-by-name .................. 496
Contents

18.4 Bohm Trees ................................................................. 505
18.5 Local structure of the call-by-value interpretation .............. 505

Notes and References for Part VI ............................. 507

Part VII: Objects and π-calculus ........................... 513

Introduction to Part VII .................................................. 515

19 Semantic Definition ....................................................... 517
19.1 A programming language .............................................. 517
19.2 Modelling examples .................................................... 522
19.3 Formal definition ......................................................... 528

20 Applications ................................................................. 533
20.1 Some properties of declarations and commands ............... 533
20.2 Proxies ................................................................. 535
20.3 An implementation technique ....................................... 539
20.4 A program transformation ............................................ 541

Notes and References for Part VII ............................. 546

List of Tables ................................................................. 548
List of Notations ............................................................. 550
Bibliography ................................................................. 562
Index ................................................................. 576
Computer science aims to explain the way computational systems behave for us. The notion of calculational process, or algorithm, is a lot older than computing technology; so, oddly enough, a lot of computer science existed before modern computers. But the invention of real stored-program computers presented enormous challenges; these tools can do a lot for us if we describe properly what we want done. So computer science has made immense strides in ways of presenting data and algorithms, in ways of manipulating these presentations themselves as data, in matching algorithm description to task description, and so on. Technology has been the catalyst in the growth of modern computer science.

The first large phase of this growth was in free-standing computer systems. Such a system might have been a single computer program, or a multi-computer serving a community by executing several single programs successively or simultaneously. Computing theorists have built many mathematical models of these systems, in relation to their purposes. One very basic such model – the $\lambda$-calculus – is remarkably useful in this role, even if it was designed by Alonzo Church around 1940.

The second phase of the growth of computer science is in response to the advent of computer networks. No longer are systems freestanding; they interact, collaborate and interrupt each other. This has an enormous effect on the way we think about our systems. We can no longer get away with considering each system as sequential, goal-directed, deterministic or hierarchical; networks are none of these. So if we confine ourselves to such concepts then we remain dumb if asked to predict whether a network will behave in a proper or an improper way; for example, whether someone logging in to his bank account (as happened recently) finds himself scanning someone else’s account instead of his own.

The present book is a rigorous account of a basic calculus which aims to underpin our theories of interactive systems, in the same way that the $\lambda$-calculus did for freestanding computation. The authors are two of the original researchers.
on the π-calculus, which is now over ten years old and has served as a focus for much theoretical and practical experiment. It cannot claim to be definitive; in fact, since it was designed it has become common to express ideas about interaction and mobility in variants of the calculus. So it has become a kind of workshop of ideas.

That’s the spirit in which the book is written. Half the book analyses the constructions of the calculus, searching out its meaning and exploring its expressivity by looking at weaker variants, or by looking at various type disciplines. Enthusiasts about types in programming will be struck to find that π-calculus types don’t just classify values; they classify patterns of behaviour. This reflects the fact that what matters most in mobile interactive systems is not values, but connectivity and mobility of processes. With or without types, the unifying feature is behaviour, and what it means to say that two different processes behave the same.

The later part of the book deals with two generic applications. One of these is classical; how the π-calculus can actually do the old job which the λ-calculus does in underpinning conventional programming. The other is modern; how the calculus informs one of the most important models of interaction, the object-oriented model. These applications bring together much of the theory developed earlier; together, they show what a small set of constructs, provided that they emphasize interaction rather than calculation, can still bring some conceptual unity to the greatly extended scope of modern computing.

This book has been a labour of love for the authors over several years. Their scholarship is immense, and their organisation of ideas meticulous. As one privileged to have worked closely with them both, it’s a great pleasure to be able to recommend the result as a storehouse of ideas and techniques which is unlikely to be equalled in the next decade or two.

Robin Milner
Cambridge
February 2001
Preface

Mobile systems, whose components communicate and change their structure, now pervade the informational world and the wider world of which it is a part. But the science of mobile systems is yet immature. This science must be developed if we are to understand mobile systems and if we are to design systems so that they do what they are intended to do. This book presents the \( \pi \)-calculus, a theory of mobile systems, and shows how to use it to express systems precisely and reason about their behaviour rigorously.

The book is intended to serve both as a reference for the theory and as an extended demonstration of how to use the \( \pi \)-calculus to express systems and analyse their properties. The book therefore presents the theory in detail, with emphasis on proof techniques. How to use the techniques is shown both in proofs of results that form part of the theory and in example applications of it.

The book is in seven Parts. Part I introduces the \( \pi \)-calculus and develops its basic theory. Part II presents variations of the basic theory and important subcalculi of the \( \pi \)-calculus. A distinctive feature of the calculus is its rich theory of types for mobile systems. Part III introduces this theory, and Part IV shows how it is useful for understanding and reasoning about systems. Part V examines the relationship between the \( \pi \)-calculus and higher-order process calculi. Part VI analyses the relationship between the \( \pi \)-calculus and the \( \lambda \)-calculus. Part VII shows how ideas from \( \pi \)-calculus can be useful in object-oriented design and programming.

The book is written at the graduate level and is intended for computer scientists in terest in mobile systems. It assumes no prior acquaintance with the \( \pi \)-calculus: both the theory and the viewpoint that underlies it are explained from the beginning.

Although the book covers quite a lot of ground, several topics, notably logics for mobility, and denotational and non-interleaving semantics, are not treated at all. The book contains detailed accounts of a selection of topics, chosen for
their interest and because they allow us to explore concepts and techniques that can also be used elsewhere. Each Part ends with some references to sources and additional notes on related topics. We have attempted the arduous task of referring to all relevant published work. The references given provide starting points for a reader who wishes to go more deeply into particular topics. Sometimes, an element of arbitrariness in the choice of references was inevitable.

Many exercises are suggested to help appreciation of the material; the more difficult of them are marked with an asterisk. We intend to maintain a Web page for general information and auxiliary material about the book. At the time of writing, this page is located at

http://www-sop.inria.fr/mimosa/personnel/Davide.Sangiorgi/
Book_pi.html

Acknowledgements Our greatest debt is to Robin Milner. The field that is the subject of this book was shaped by his fundamental work on CCS and in creating and developing the π-calculus. Further, we have been privileged to have worked with Milner, and his influence on our approach to the subject and how to write about it are profound.

We thank the many colleagues – too many to mention here – with whom we have worked on or discussed π-calculus and related topics, and whose insights and comments have contributed to our understanding.

We are grateful to the following people for reading parts of a draft of the book and offering comments that helped us improve it: Michael Baldamus, Silvia Crafa, Cédric Fournet, Daniel Hirschkoff, Kohei Honda, Naoki Kobayashi, Giovanni Lagorio, Cédric Lhuissaine, Huimin Lin, Barbara König, Robin Milner, Julian Rathke, Vasco Vasconcelos, Nobuko Yoshida, and especially Marco Pistore.

We record our appreciation of the work of David Tranah and his colleagues at Cambridge University Press in guiding the book into print.

Finally, we thank Laurence Sangiorgi and Katharine Grevling for their encouragement, assistance, and patience during the seemingly interminable process of writing.