# communicating and mobile systems: the π-calculus

Robin Milner

TN929.5
M659

# Communicating and Mobile Systems: the $\pi$-Calculus

ROBIN MILNER

*Computer Laboratory, University of Cambridge*

CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK http://www.cup.cam.ac.uk
40 West 20th Street, New York, NY 10011-4211, USA http://www.cup.org
10 Stamford Road, Oakleigh, Melbourne 3166, Australia

First published 1999

Printed in the United Kingdom at the University Press, Cambridge

Typeset in Times 10/13pt, in LaTeX $2_\varepsilon$ [EPC]

*A catalogue record of this book is available from the British Library*

*Library of Congress Cataloguing in Publication data*

Milner, R. (Robin). 1934–
Communicating and mobile systems : the $\pi$-calculus / Robin Milner.
p. cm.
ISBN 0 521 64320 1 (hc.). – ISBN 0 521 65869 1 (pbk.).
1. Mobile computing. 2. Telecommunication systems.
3. Pi-calculus. I. Title.
QA76.59.M55 1999
004.6′2–dc21 98-39479 CIP

ISBN 0 521 64320 1 hardback
ISBN 0 521 65869 1 paperback

**Communicating and Mobile Systems: the $\pi$-Calculus**

Communication is a fundamental and integral part of computing, whether between different computers on a network, or between components within a single computer. In this book Robin Milner introduces a new way of modelling communication that reflects its position. He treats computers and their programs as themselves built from communicating parts, rather than adding communication as an extra level of activity. Everything is introduced by means of examples, such as mobile phones, job schedulers, vending machines, data structures, and the objects of object-oriented programming. But the aim of the book is to develop a theory, the $\pi$-calculus, in which these things can be treated rigorously.

The $\pi$-calculus differs from other models of communicating behaviour mainly in its treatment of mobility. The movement of a piece of data inside a computer program is treated exactly the same as the transfer of a message – or indeed an entire computer program – across the internet. One can also describe networks which reconfigure themselves.

The calculus is very simple but powerful. Its most prominent notion is that of a name, and it has two important ingredients: the concept of behavioural (or observational) equivalence, and the use of a new theory of types to classify patterns of interactive behaviour. The internet, and its communication protocols, fall within the scope of the theory just as much as computer programs, data structures, algorithms and programming languages.

This book is the first textbook of the subject; it has been long-awaited by professionals and will be welcomed by them, and their students.

BASIC CONSTRUCTIONS

| | | |
|---|---|---|
| $A(\vec{a}) \overset{\text{def}}{=} P_A$ | process definition | 3.4 |
| $\Sigma\alpha_i.P_i$ | summation | 3.4 |
| $A\langle a_1, \ldots, a_n\rangle$ | process instance | 3.4 |
| $\{\vec{b}/\vec{a}\}P$ | substitution | 3.4 |
| $P \mid Q$ | composition | 4.3 |
| $\mathsf{new}\, a\, P$ | restriction | 4.3 |
| $P \frown Q$ | linking | 4.4 |

EQUIVALENCES

| | | |
|---|---|---|
| $\equiv$ | structural congruence | 4.4 |
| $\sim$ | strong equivalence | 3.3 |
| $\approx$ | weak equivalence | 6.2 |

$\pi$-CALCULUS CONSTRUCTIONS

| | | |
|---|---|---|
| $x(y)$ , $\overline{x}\langle y\rangle$ | action prefix (monadic) | 9.1 |
| $\Sigma\pi_i.P_i$ | summation | 9.1 |
| $!P$ | replication | 9.1 |
| $x(\vec{y})$ , $\overline{x}\langle\vec{y}\rangle$ | action prefix (polyadic) | 9.4 |
| $F;G$ | sequential composition | 9.6 |
| $(\vec{x}).P$ | abstraction | 12.1 |
| $\mathsf{new}\,\vec{x}\,\langle\vec{y}\rangle.P$ | concretion | 12.1 |
| $F@C$ | application | 12.1 |
| $\Sigma\alpha_i A_i$ | summation of agents | 12.1 |

# Preface

Over the last thirty years or so, computer science has seriously taken up the challenge to understand the behaviour of communicating systems in the same way as it understands the behaviour of computer programs.

There is little pre-existing theory which can help. This is perhaps surprising, because the theory of computing has developed over a very long period as a part of mathematics and logic, and indeed it influenced the design of early stored-program computers. By comparison, a theory of communication as a smooth extension of programming is in its adolescence.

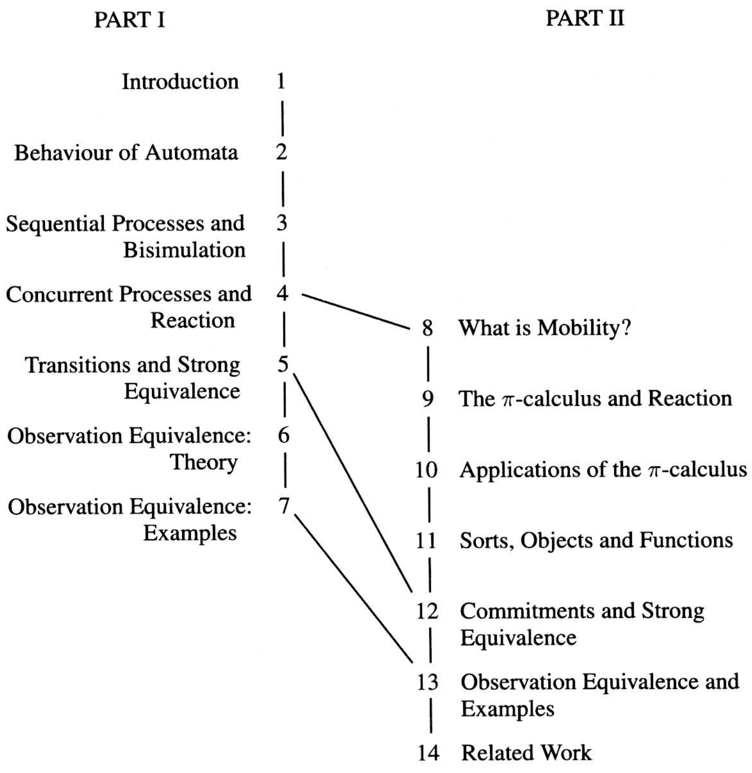
But theories usually arise to explain practice. Recently there has been a sea-change in computing practice; due to techological advances interactive systems are becoming the norm rather than the exception, and our whole view of computing has changed correspondingly. The new technology has created the need to expand our theory of sequential algorithmic processes to systems where interaction plays a significant and even dominant rôle.

One of the most challenging developments, both technically and conceptually, is the advent of mobile computing. People, computers and software now continually move among each other; moreover, some of the movement is physical and some (e.g. the movement of links) is virtual. As we experience this, we must somehow distil basic ideas which will help us to create reliable mobile systems which do what we want them to do.

Analysing the behaviour of mobile systems at the design stage is much harder than it ever was for sequential computer programs. This is partly because we lack ways even to express such behaviour accurately, in order to specify what must be designed. The $\pi$-calculus was developed in the late 1980s with just this goal in mind; this book introduces it with motivation and examples, but also with mathematical precision.

**Who should read the book** The book has grown out of a lecture course of sixteen lectures to final-year undergraduate students at Cambridge. It is designed for such a course. In making the book from the lecture notes I have resisted adding more material; I have only added explanations. The material is challenging for undergraduates; the book can also be used as a basis for graduate courses.

**How to read the book** The book divides clearly into two parts. Part I deals with interactive systems which are not mobile, and represents a self-contained review of previous work on CCS (a Calculus of Communicating Systems) [9, 10]. Part II introduces mobility, in the form of the dynamic creation of new links between active processes. But one need not read all of Part I before Part II. The diagram below shows the dependency of chapters.


PART I
PART II
Introduction 1
Behaviour of Automata 2
Sequential Processes and Bisimulation 3
Concurrent Processes and Reaction 4
Transitions and Strong Equivalence 5
Observation Equivalence: Theory 6
Observation Equivalence: Examples 7
8 What is Mobility?
9 The π-calculus and Reaction
10 Applications of the π-calculus
11 Sorts, Objects and Functions
12 Commitments and Strong Equivalence
13 Observation Equivalence and Examples
14 Related Work

There are many paths through some or all of the chapters:

- Part I, by itself is a good introduction to the algebraic treatment of communicating systems; it emphasizes the kind of theoretical problem which arises with concurrency, but preserves a balance between theory and examples.

- Chapters 1–4 and 8–11 make a good introduction to mobile interactive systems, with emphasis on applications and less upon the behavioural theory. The examples of Chapter 7 can be added to this diet; they can be appreciated to a reasonable extent without the preparation of Chapters 5 and 6.
- Chapters 1–5 and 8–12 make a coherent whole, dealing with everything except the concept of observation equivalence. Chapters 6, 7 and 13 can then be tackled together.

Thus the mixture of theory and examples can be varied to taste. The theorems are important, but very often the practical applications can be appreciated without them.

**Acknowledgements** I would first like to thank the students who have helped me over the last three years to write down these ideas in progressively better form, and particularly those who suffered the earlier attempts. It is a continual source of excitement to me that to teach a new subject is so important for one's understanding of it. The ideas in the $\pi$-calculus are due in great part to Mogens Nielsen and Uffe Engberg who took important steps towards it, to Joachim Parrow and David Walker who first worked the calculus out in detail with me, and to Davide Sangiorgi who made important subsequent advances. Alexis Donnelly, Peter Sewell and David Walker have read earlier drafts of the book in considerable detail and made valuable suggestions.

# Contents

*Glossary* *page* viii
*Preface* x

**Part I: Communicating Systems** 1

**1 Introduction** 3

**2 Behaviour of Automata** 8
2.1 Automata 8
2.2 Regular sets 10
2.3 The language of an automaton 11
2.4 Determinism versus nondeterminism 12
2.5 Black boxes, or reactive systems 13
2.6 Summary 15

**3 Sequential Processes and Bisimulation** 16
3.1 Labelled transition systems 16
3.2 Strong simulation 17
3.3 Strong bisimulation 18
3.4 Sequential process expressions 20
3.5 Boolean buffer 22
3.6 Scheduler 23
3.7 Counter 24
3.8 Summary 25

**4 Concurrent Processes and Reaction** 26
4.1 Labels and flowgraphs 26
4.2 Observations and reactions 27
4.3 Concurrent process expressions 29
4.4 Structural congruence 31

4.5 Reaction rules 33
4.6 Summary 37

**5 Transitions and Strong Equivalence 38**
5.1 Labelled transitions 38
5.2 Strong bisimilarity and applications 45
5.3 Algebraic properties of strong equivalence 48
5.4 Congruence 50
5.5 Summary 51

**6 Observation Equivalence: Theory 52**
6.1 Observations 52
6.2 Weak bisimulation 53
6.3 Unique solution of equations 58
6.4 Summary 59

**7 Observation Equivalence: Examples 60**
7.1 Lottery 60
7.2 Job Shop 61
7.3 Scheduler 64
7.4 Buffer 67
7.5 Stack and Counter 69
7.6 Discussion 73

**Part II: The $\pi$-Calculus 75**

**8 What is Mobility? 77**
8.1 Limited mobility 79
8.2 Mobile phones 80
8.3 Other examples of mobility 83
8.4 Summary 86

**9 The $\pi$-Calculus and Reaction 87**
9.1 Names, actions and processes 87
9.2 Structural congruence and reaction 89
9.3 Mobility 91
9.4 The polyadic $\pi$-calculus 93
9.5 Recursive definitions 94
9.6 Abstractions 96
9.7 Summary 97

**10 Applications of the $\pi$-Calculus 98**
10.1 Simple systems 98
10.2 Unique handling 100
10.3 Data revisited 103

| | | |
|---|---|---|
| 10.4 | Programming with lists | 106 |
| 10.5 | Persistent and mutable data | 109 |
| **11** | **Sorts, Objects, and Functions** | 113 |
| 11.1 | A hierarchy of channel types? | 113 |
| 11.2 | Sorts and sortings | 114 |
| 11.3 | Extending the sort language | 116 |
| 11.4 | Object-oriented programming | 119 |
| 11.5 | Processes and abstractions as messages | 123 |
| 11.6 | Functional computing as name-passing | 125 |
| **12** | **Commitments and Strong Bisimulation** | 129 |
| 12.1 | Abstractions and concretions | 129 |
| 12.2 | Commitment rules | 132 |
| 12.3 | Strong bisimulation, strong equivalence | 134 |
| 12.4 | Congruence | 136 |
| 12.5 | Basic congruence properties of replication | 138 |
| 12.6 | Replicated resources | 140 |
| 12.7 | Summary | 141 |
| **13** | **Observation Equivalence and Examples** | 142 |
| 13.1 | Experiments | 142 |
| 13.2 | Weak bisimulation and congruence | 143 |
| 13.3 | Unique solution of equations | 145 |
| 13.4 | List programming | 146 |
| 13.5 | Imperative programming | 147 |
| 13.6 | Elastic buffer | 148 |
| 13.7 | Reduction in the $\lambda$-calculus | 151 |
| **14** | **Discussion and related work** | 153 |
| *References* | | 157 |
| *Index* | | 159 |

# Glossary

The important notations, with the section number of their first appearance.

| ENTITY SET | ENTITY | DESCRIPTION | |
|---|---|---|---|
| $\mathcal{N}$ | $a, \ldots, x, \ldots$ | names | 3.1 |
| $\overline{\mathcal{N}}$ | $\overline{a}, \ldots, \overline{x}, \ldots$ | co-names | 3.1 |
| $\mathcal{L}$ | $\lambda$ | labels | 3.1 |
| $\mathcal{Q}$ | $p, q, \ldots$ | states | 3.1 |
| | $\mathcal{R}, \mathcal{S}, \ldots$ | simulation, bisimulation | 3.2 |
| $\mathcal{P}^{\mathrm{seq}}$ | $P, Q, \ldots$ | sequential processes | 3.4 |
| *Act* | $\alpha, \beta, \ldots$ | actions | 4.2 |
| | $\tau$ | silent action | 4.2 |
| $\mathcal{P}$ | $P, Q, \ldots$ | concurrent processes | 4.3 |
| | $\mathcal{C}$ | process contexts | 4.4 |
| $\mathcal{P}^{\pi}$ | $P, Q, \ldots$ | $\pi$-calculus processes | 9.1 |
| | $\pi$ | action prefixes | 9.1 |
| $\Sigma$ | $\sigma$ | sorts | 11.2 |
| $\Gamma$ | $C$ | sort constructors | 11.3 |
| | $F, G, \ldots$ | abstractions | 12.1 |
| | $C, D, \ldots$ | concretions | 12.1 |
| $\mathcal{A}^{\pi}$ | $A, B, \ldots$ | $\pi$-calculus agents | 12.1 |

| ACTION RELATIONS | | |
|---|---|---|
| $\xrightarrow{\alpha}$ | labelled transition | 3.1 |
| $\xrightarrow{\alpha}$ | commitment | 12.2 |
| $\longrightarrow$ | reaction | 4.5 |
| $\Rightarrow$ | empty experiment | 6.1 |
| $\overset{e}{\Rightarrow}$ | experiment | 6.1 |

# Part I

## Communicating Systems

# 1

# Introduction

This book introduces a calculus for analysing properties of concurrent communicating processes, which may grow and shrink and move about.

Building communicating systems is not a well-established science, or even a stable craft; we do not have an agreed repertoire of constructions for building and expressing interactive systems, in the way that we (more-or-less) have for building sequential computer programs.

But nowadays most computing involves interaction – and therefore involves systems with components which are concurrently active. Computer science must therefore rise to the challenge of defining an underlying model, with a small number of basic concepts, in terms of which *interactional* behaviour can be rigorously described.

The same thing was done for *computational* behaviour a long time ago; logicians came up with Turing machines, register machines (on which imperative programming languages are built) and the lambda calculus (on which the notion of parametric procedure is founded). None of these models is concerned with interaction, as we would normally understand the term. Their basic activity consists of reading or writing on a storage medium (tape or registers), or invoking a procedure with actual parameters. Instead, we shall work with a model whose basic action is to communicate across an interface with a *handshake*, which means that the two participants synchronize this action.

Let us think about some simple examples of processes which do this handshaking. They can be physical or virtual, hardware or software. As a very physical system, consider a vending machine e.g. for selling drinks. It has links with its environment: the slot for money, the drink-selection buttons, the button for getting your change, the delivery point for a drink. The machine's pattern of interaction at these links is not entirely trivial – as we shall see in Chapter 2.

Physical systems tend to have permanent physical links; they have *fixed* structure. But most systems in the informatic world are not physical; their links may be virtual or symbolic. An obvious modern example is the linkage among agents on the internet or worldwide web. When you click on a symbolic link on your screen, you induce a handshake between a local process (your screen agent) and a remote process. These symbolic links can also be created or destroyed on the fly, by you and others. Virtual links can also consist of radio connection; consider the linkage between planes and the control tower in an air-traffic control system. Systems like these, with transient links, have *mobile* structure. In Chapter 8 we shall look at a very simple example involving mobile telephones.

We do not normally think of vending machines or mobile phones as doing computation, but they share the notion of interaction with modern distributed computing systems. This common notion underlies a theory of a huge range of modern informatic systems, whether computational or not. This is the theory we shall develop.

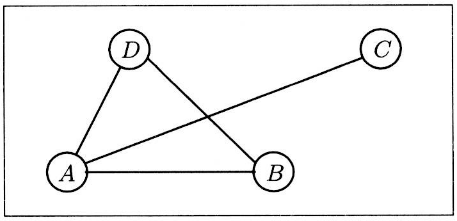
This book is not about design; for example, it will not teach you how best to design a concurrent operating system. Instead, we shall try to isolate concepts which allow designers to think clearly, not only when analysing interactive systems but even when expressing their designs in the first place. So we shall proceed with the help of examples – not large systems, but small ones illustrating key notions and problems.

A central question we shall try to answer is: when do two interactive systems have equivalent behaviour, in the sense that we can unplug one and plug in the other – in any environment – and not tell the difference? This is a theoretical question, but vitally important in practice. Until we know what constitutes similarity or difference of behaviour, we cannot claim to know what 'behaviour' *means* – and if that is the case then we have no precise way of explaining what our systems do!

Therefore our theory will focus on equivalence of behaviour. In fact we use this notion as a means of specifying how a designed system should behave; the designed system is held to be correct if its actual behaviour is equivalent to the specified behaviour. Chapters 7 and 13 contain several examples of how to prove such behavioural equivalence.

We shall begin at a familiar place, the classical theory of automata. We shall then extend these automata to allow them to run concurrently and to interact – which they will do by synchronizing their transitions from one state to another. This allows us to consider each system component, whether elementary or containing subcomponents, as an automaton.

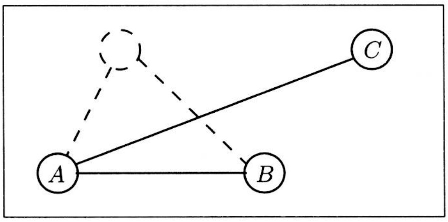
For such systems of interacting automata we shall find it useful to represent their interconnection by diagrams, such as the following:



(1)

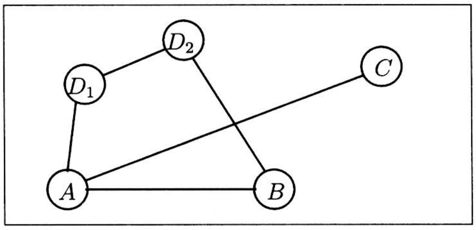
Here, an arc between two component automata $A$ and $B$ of a system means that they *may* interact – that is, $A$ and $B$ may sometimes synchronize their state transitions.

In many systems this linkage, or spatial structure, remains fixed as the system's behaviour unfolds. But in certain applications the spatial structure may *evolve*; for example the component $D$ may *die* (1→2):



(2)

or may *divide* into two components (1→3):



(3)

This mode of evolution covers a large variety of behaviour. For example, in understanding a high-level programming language one can treat each activation of a recursive procedure as an system component, whose lifetime lasts from a call of the procedure to a return from it; this extends smoothly to the case in which concurrent activations of the same procedure are allowed. Again, a communication handler may under certain conditions create a 'subagent' to deal with certain transactions; the subagent will carry out certain delegated interactions, and die when its task is done.

A calculus called CCS (Calculus of Communicating Systems) was devel-