

# Preface

Probabilistic techniques in computer programs and systems are becoming more and more widely used, for increased efficiency (as in random algorithms), for symmetry breaking (distributed systems) or as an unavoidable artefact of applications (modelling fault-tolerance). Because interest in them has been growing so strongly, stimulated by their many potential uses, there has been a corresponding increase in the study of their correctness — for the more widespread they become, the more we will depend on understanding their behaviour, and their limits, exactly.

In this volume we address that last concern, of understanding: we present a method for *rigorous reasoning about probabilistic programs and systems*. It provides an operational model — “how they work” — and an associated program logic — “how we should reason about them” — that are designed to fit together. The technique is simple in principle, and we hope that with it we will be able to increase dramatically the effectiveness of our analysis and use of probabilistic techniques in practice.

Our contribution is a probabilistic calculus that operates at the level of the program text, and it is *light-weight* in the sense that the amount of reasoning is similar in size and style to what standard assertional techniques require. In the fragment at right, for example, each potential loop entry occurs with probability  $1/2$ ; the resulting iteration establishes  $x \geq 1/2$  with probability exactly  $p$  for any  $0 \leq p \leq 1$ . It is thus an implementation of the general operation *choose with probability  $p$* , but it uses only simple tests of unbiased random bits (to implement the loop guard). It should take only a little quantitative logic to confirm that claim, and indeed we will show that just four lines of reasoning suffice.

```
x := p;  
while 1/2 do  
  x := 2x;  
  if x ≥ 1  
    then x := x - 1  
  fi  
od
```

Economy and precision of reasoning are what we have come to expect for standard programs; there is no reason we should accept less when they are probabilistic.

---

*The cover illustration comes from page 59.  
The program fragment is adapted from Fig. 7.7.10 on page 210.*

*Scope and applicability*

Methods for the analysis of probabilistic systems include automata, labelled transition systems, model checking and logic (*e.g.* dynamic or temporal). Our work falls into the last category: we overlay the Hoare-logic paradigm with probabilistic features imported from Markov processes, taking from each the essential characteristics required for a sound mathematical theory of refinement and proof. The aim is to accommodate modelling and analysis of both sequential and distributed probabilistic systems, and to allow — even encourage — movement between different levels of abstraction.

Our decision to focus on *logic* — and a proof system for it — was motivated by our experience with logical techniques more generally: they impose a discipline and order which promotes clarity in specifications and design; the resulting proofs can often be carried out, and checked, with astonishing conciseness and accuracy; and the calculation rules of the logic lead to an algebra that captures useful equalities and inequalities at the level of the programs themselves.

Although we rely ultimately on an operational model, we use it principally to validate the logic (and that, in turn, justifies the algebra) — direct reliance on the model’s details for individual programs is avoided if possible. (However we do not hesitate to use such details to support our intuition.) We feel that operational reasoning is more suited to the algorithmic methods of verification used by model checkers and simulation tools which can, for specific programs, answer questions that are impractical for the general approach that a logic provides.

Thus the impact of our approach is most compelling when applied to programs which are intricate either in their implementation or their design, or have generic features such as undetermined size or other parameters. They might appear as probabilistic source-level portions of large sequential programs, or as abstractions from the probabilistic modules of a comprehensive system-level design; we provide specific examples of both situations. In the latter case the ability to abstract modules’ properties has a significant effect on the overall verification enterprise.

*Technical features*

Because we generalise the well-established assertional techniques of specifications, pre- and postconditions, there is a natural continuity of reasoning style evident in the simultaneous use of the new and the familiar approaches: the probabilistic analysis can be deployed more, or less, as the situation warrants.

A major feature is that we place probabilistic choice and abstraction together, in the same framework, without having to factor either of them out for separate treatment unless we wish to (as in fact we do in Chap. 11). This justifies the *abstraction and refinement* of our title, and is what gives

us access to the stepwise-development paradigm of standard programming where systems are “refined” from high levels of abstraction towards the low levels that include implementation detail.

As a side-effect of including abstraction, we retain its operational counterpart *demonic choice* as an explicit operator  $\sqcap$  in the cut-down probabilistic programming language *pGCL* which we use to describe our algorithms — that is, the new probabilistic choice operator  $_p\oplus$  refines demonic choice rather than replacing it. In Chap. 8 we consider angelic choice  $\sqcup$  as well, which is thus a further refinement.

Probabilistic and demonic choice together allow an elementary treatment of the hybrid that selects “with probability *at least*  $p$ ” (or similarly “*at most*  $p$ ”), an abstraction which accurately models our unavoidable ignorance of exact probabilities in real applications. Thus in our mathematical model we are able to side-step the issue of “approximate refinement.”

That is, rather than saying “this coin refines a fair coin with probability 95%,” we would say “this coin refines one which is within 5% of being fair.” This continues the simple view that either an implementation refines a specification or it does not, which simplicity is possible because we have retained the original treatment in terms of sets of behaviours: abstraction is inclusion; refinement is reverse inclusion; and demonic choice is union. In that way we maintain the important relationship between the three concepts. (Section 6.5 on pp. 169ff illustrates this geometrically.)

### *Organisation and intended readership*

The material is divided into three major parts of increasing specialisation, each of which can to a large extent be studied on its own; a fourth part contains appendices. We include a comprehensive index and extensive cross-referencing.

Definitions of notation and explanations of standard mathematical techniques are carefully given, rather than simply assumed; they appear as footnotes at their first point of use and are made visually conspicuous by using SMALL CAPITALS for the defined terms (where grammar allows). Thus in many cases a glance should be sufficient to determine whether any footnote contains a definition. In any case all definitions, whether or not in footnotes, may be retrieved by name through the index; and those with numbers are listed in order at page xvii.

Because much of the background material is separated from the main text, the need for more advanced readers to break out of the narrative should be reduced. We suggest that on first reading it is better to consult the footnotes only when there is a term that appears to require definition — otherwise the many cross-references they contain may prove distracting, as they are designed for “non-linear” browsing once the main ideas have already been assimilated.

Part I, *Probabilistic guarded commands*, gives enough introduction to the probabilistic logic to prove properties of small programs such as the one earlier, for example at the level of an undergraduate course for Formal-Methods-inclined students that explains “what to do” but not necessarily “why it is correct to do that.” These would be people who need to understand how to reason about programs (and why), but would see the techniques as intellectual tools rather than as objects of study in their own right.

We have included many small examples to serve as models for the approach (they are indexed under *Programs*), and there are several larger case studies (for example in Chap. 3).

Part II, *Semantic structures*, develops in detail the mathematics on which the probabilistic logic is built and with which it is justified. That is, whereas the earlier sections present and illustrate the new reasoning techniques, this part shows where they have come from, why they have the form they do and — crucially — why they are correct.

That last point is especially important for students intending to do research in logic and semantics, as it provides a detailed and extended worked example of the fundamental issue of proving reasoning techniques *themselves* to be correct (more accurately, “valid”), a higher-order concept than the more familiar theme of the previous part in which we presented the techniques *ex cathedra* and used them to verify particular programs.

This part would thus be suitable for an advanced final-year undergraduate or first-year graduate course, and would fit in well with other material on programming semantics. It defines and illustrates the use of many of the standard tools of the subject: lattices, approximation orders, fixed points, semantic injections and retractions *etc.*

Part III, *Advanced topics*, concentrates on more exotic methods of specification and design, in this case probabilistic temporal/modal logics. Its final chapter, for example, contains material only recently discovered and leads directly into an up-to-date research area. It would be suitable for graduate students as an introduction to this specialised research community.

Part IV includes appendices collecting material that either leads away from the main exposition — *e.g.* alternative approaches and why we have not taken them — or supports the text at a deeper level, such as some of the more detailed proofs.

It also contains a short list of algebraic laws that demonic/probabilistic program fragments satisfy, generated mainly by our needs in the examples and proofs of earlier sections. An interesting research topic would be a more systematic elaboration of that list with a view to incorporating it into probabilistic Kleene- or omega algebras for distributed computations.

Overall, readers seeking an introduction to probabilistic formal methods could follow the material in order from the beginning. Those with more experience might instead sample the first chapter from each part, which would give an indication of the scope and flavour of the approach generally.

### *Original sources*

Much of the material is based on published research, done with our colleagues, in conference proceedings and journal articles; but here it has been substantially updated and rationalised — and we have done our best to bring the almost ten years’ worth of developing notation into a uniform state.

For self-contained presentations of the separate topics, and extra background, readers could consult our earlier publications as shown overleaf.

At the end of each chapter we survey the way in which our ideas have been influenced by — and in some cases adopted from — the work of other researchers, and we indicate some up-to-date developments.

### *Acknowledgements*

Our work on probabilistic models and logic was carried out initially at the University of Oxford, together with Jeff Sanders and Karen Seidel and with the support of the UK’s *Engineering and Physical Sciences Research Council* (the EPSRC) during two projects led by Sanders and Morgan over the years 1994–2001.

Morgan spent sabbatical semesters in 1995–6 at the University of Utrecht, as the guest of S. Doaitse Swierstra, and at the University of Queensland and the Software Verification and Research Centre (SVRC), as the guest of David Carrington and Ian Hayes. The foundational work the EPSRC projects produced during that period — sometimes across great distances — benefited from the financial support of those institutions but especially from the academic environment provided by the hosts and by the other researchers who were receptive to our initial ideas [MMS96].

Ralph Back at Åbo Akademi hosted our group’s visit to Turku for a week in 1996 during which we were able to explore our common interests in refinement and abstraction as it applied to the new domain; that led later to a three-month visit by Elena Troubitsyna from the *Turku Center for Computer Science* (TUCS), to our group in Oxford in 1997, and contributed to what has become Chap. 4 [MMT98].

David Harel was our host for a two-week visit to Israel in 1996, during which we presented our ideas and benefited from the interaction with researchers there.

### Chapters' dependence on original sources

Chapter 1	<i>see</i>	[MM99b, SMM, MMS00]
Chapter 2	<i>see</i>	[Mor96, MMS00]
Chapter 3	<i>see</i>	[MM99b]
Chapter 4	<i>see</i>	[MMT98]
Chapter 5	<i>see</i>	[MMS96]
Chapter 6		<i>is new material</i>
Chapter 7	<i>see</i>	[Mor96, MM01b]
Chapter 8	<i>see</i>	[MM01a]
Chapter 9	<i>see</i>	[MM97]
Chapter 10	<i>see</i>	[MM99a]
Chapter 11	<i>see</i>	[MM02]

The sources listed opposite are in chronological order of writing, thus giving roughly the logical evolution of the ideas.

Subsequently we have continued to work with Sanders and with Ken Robinson, Thai Son Hoang and Zhendong Jin, supported by the *Australian Research Council* (ARC) over the (coming) years 2001–8 in their *Large Grant* and *Discovery* programmes, at the Universities of Macquarie and of New South Wales.

Joe Hurd from the Computer Laboratory at Cambridge University visited us in 2002, with financial assistance from Macquarie University; and Orieta Celiku was supported by TUCS when she visited in 2003. Both worked under McIver's direction on the formalisation of *pGCL*, and its logic, in the mechanised logic *HOL*.

Hoang, Jin and especially Eric Martin have helped us considerably with their detailed comments on the typescript; also Ralph Back, Ian Hayes, Michael Huth, Quentin Miller and Wayne Wheeler have given us good advice. Section B.1 on the algebraic laws satisfied by probabilistic programs has been stimulated by the work (and the critical eyes) of Steve Schneider and his colleagues at Royal Holloway College in the U.K.

We thank the members of IFIP Working Groups 2.1 and 2.3 for their many comments and suggestions.

*LRI* Paris  
May 2004

Annabelle McIver  
Carroll Morgan

---

*In memoriam* AJMvG

## List of sources in order of writing

- [MMS96] C.C. Morgan, A.K. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–53, May 1996.
- [Mor96] C.C. Morgan. Proof rules for probabilistic loops. In He Jifeng, John Cooke, and Peter Wallis, editors, *Proceedings of the BCS-FACS 7th Refinement Workshop*, Workshops in Computing. Springer-Verlag, July 1996.
- [MM01b] A.K. McIver and C.C. Morgan. Partial correctness for probabilistic programs. *Theoretical Computer Science*, 266(1–2):513–41, 2001.
- [SMM] K. Seidel, C.C. Morgan, and A.K. McIver. Probabilistic imperative programming: a rigorous approach. Extended abstract appears in Groves and Reeves [GR97], pages 1–2.
- [MMT98] A.K. McIver, C.C. Morgan, and E. Troubitsyna. The probabilistic steam boiler: a case study in probabilistic data refinement. In J. Grundy, M. Schwenke, and T. Vickers, editors, *Proc. International Refinement Workshop, ANU, Canberra*, Discrete Mathematics and Computer Science, pages 250–65. Springer-Verlag, 1998.
- [MM01a] A.K. McIver and C.C. Morgan. Demonic, angelic and unbounded probabilistic choices in sequential programs. *Acta Informatica*, 37:329–54, 2001.
- [MM99b] C.C. Morgan and A.K. McIver. *pGCL*: Formal reasoning for random algorithms. *South African Computer Journal*, 22, March 1999.
- [MM97] C.C. Morgan and A.K. McIver. A probabilistic temporal calculus based on expectations. In Groves and Reeves [GR97], pages 4–22.
- [MM99a] C.C. Morgan and A.K. McIver. An expectation-based model for probabilistic temporal logic. *Logic Journal of the IGPL*, 7(6):779–804, 1999.
- [MMS00] A.K. McIver, C.C. Morgan, and J.W. Sanders. Probably Hoare? Hoare probably! In J.W. Davies, A.W. Roscoe, and J.C.P. Woodcock, editors, *Millennial Perspectives in Computer Science*, Cornerstones of Computing, pages 271–82. Palgrave, 2000.
- [MM02] A.K. McIver and C.C. Morgan. Games, probability and the quantitative  $\mu$ -calculus qMu. In *Proc. LPAR*, volume 2514 of *LNAI*, pages 292–310. Springer-Verlag, 2002.

Abstraction, Refinement and Proof for Probabilistic  
Systems

McIver, A.; Morgan, C.C.

2005, XX, 388 p. 63 illus., Hardcover

ISBN: 978-0-387-40115-7