Yuxin Deng
邓玉欣 著

Semantics of Probabilistic Processes

An Operational Approach

概率进程语义（英文版）

上海交通大學出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

Springer

国家科学技术学术著作出版基金资助出版

Semantics of Probabilistic Processes

概率进程语义

(英文版)

Yuxin Deng

邓玉欣 著

上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

Springer

图书在版编目(CIP)数据

概率进程语义:英文/邓玉欣著. —上海:上海交通大学出版社,2015

ISBN 978-7-313-13821-7

Ⅰ.①概… Ⅱ.①邓… Ⅲ.①概率—计算模型—英文 Ⅳ.①O211.1

中国版本图书馆 CIP 数据核字(2015)第 235215 号

概率进程语义(英文版)

著　　者:邓玉欣
出版发行:上海交通大学出版社　　地　　址:上海市番禺路 951 号
邮政编码:200030　　电　　话:021-64071208
出 版 人:韩建民
印　　制:苏州市越洋印刷有限公司　　经　　销:全国新华书店
开　　本:710mm×1000mm 1/16　　印　　张:16.25
字　　数:506 千字
版　　次:2015 年 11 月第 1 版　　印　　次:2015 年 11 月第 1 次印刷
书　　号:ISBN 978-7-313-13821-7/O
定　　价:88.00 元

Preface

Probabilistic concurrency theory aims to specify and analyse quantitative behaviour of concurrent systems, which necessarily builds on solid semantic foundations of probabilistic processes. This book adopts an operational approach to describing the behaviour of nondeterministic and probabilistic processes, and that the semantic comparison of different systems is based on appropriate behavioural relations such as bisimulation equivalence and testing preorders.

It mainly consists of two parts. The first part provides an elementary account of bisimulation semantics for probabilistic processes from metric, logical and algorithmic perspectives. The second part sets up a general testing framework and specialises it to probabilistic processes with nondeterministic behaviour. The resulting testing semantics is treated in depth. A few variants of it are shown to coincide, and they can be characterised in terms of modal logics and coinductively defined simulation relations. Although in the traditional (nonprobabilistic) setting, simulation semantics is in general finer than testing semantics because it distinguishes more processes for a large class of probabilistic processes, the gap between simulation and testing semantics disappears. Therefore, in this case, we have a semantics where both negative and positive results can be easily proved: to show that two processes are not related in the semantics, we just give a witness test, and to prove that two processes are related, we only need to establish a simulation relation.

While most of the results have been announced before, they are spread over several papers in the period from 2007 to 2014, and sometimes with different terminology and notation. This prevents us from having a comprehensive understanding of the bisimulation and testing semantics of probabilistic processes. In order to improve the situation, the current work brings all the related concepts and proof techniques to form a coherent and self-contained text.

Besides presenting the recent research advances in probabilistic concurrency theory, the book exemplifies the use of many mathematical techniques to solve problems in Computer Science, which is intended to be accessible to postgraduate students in Computer Science and Mathematics. It can also be used by researchers and practitioners either for advanced study or for technical reference. The reader is assumed to have some basic knowledge in discrete mathematics. Familiarity with real analysis is not a prerequisite, but would be helpful.

Most of the work reported in this book was carried out during the last few years with a number of colleagues. The testing semantics for probabilistic processes was developed in conjunction with Rob van Glabbeek, Matthew Hennessy, Carroll Morgan and Chenyi Zhang. The various characterisations of probabilistic bisimulation in Chap. 3 are based on joint work with Wenjie Du.

The BASICS laboratory at Shanghai Jiao Tong University has offered a creative and pleasant working atmosphere. Therefore, I would like to express my gratitude to Yuxi Fu and all other members of the laboratory. Thanks go also to Barry Jay, Matthew Hennessy and Carroll Morgan for having read parts of the first draft and provided useful feedback. My research on probabilistic concurrency theory has been sponsored by the National Natural Science Foundation of China under grants 61173033 and 61261130589, as well as ANR 12IS02001 "PACE".

Finally, my special gratitude goes to my family, for their unfailing support.

Shanghai
September, 2014

Yuxin Deng

List of Symbols

Contents

Chapter 1
Introduction

Abstract This introduction briefly reviews the history of probabilistic concurrency theory and three approaches to the semantics of concurrent systems: denotational, axiomatic and operational. This book focuses on the last one and more specifically on (bi)simulation semantics and testing semantics. The second section surveys the contents and main results for other chapters of the book.

Keywords Probabilistic concurrency theory · Semantics · Bisimulation · Testing

1.1 Background

Computer science aims to explain in a rigorous way how computational systems should behave, and then to design them so that they do behave as expected. Nowadays the notion of computational systems includes not only *sequential systems* but also *concurrent systems*. The attention of computer scientists goes beyond single programs in free-standing computers. For example, computer networks, particles in physics and even proteins in biology can all be considered as concurrent systems. Some classical mathematical models (e.g. the λ-calculus [1]) are successful for describing sequential systems, but they turn out to be insufficient for reasoning about concurrent systems, because what is more important now is how different components of a system interact with each other rather than their input–output behaviour.

In the 1980s *process calculi* (sometimes also called *process algebras*), notably calculus of communicating systems (CCS) [2], communicating sequential processes (CSP) [3] and algebra of communicating processes (ACP) [4, 5], were proposed for describing and analysing concurrent systems. All of them were designed around the central idea of *interaction* between processes. In those formalisms, complex systems are built from simple subcomponents, using a small set of primitive operators such as *prefix, nondeterministic choice, restriction, parallel composition* and *recursion*. Those traditional process calculi were designed to specify and verify *qualitative behaviour* of concurrent systems.

Since the 1990s, there has been a trend to study the *quantitative behaviour* of concurrent systems. Many probabilistic algorithms have been developed in order to

gain efficiency or to solve problems that are otherwise impossible to solve by deterministic algorithms. For instance, probabilities are introduced to break symmetry in distributed coordination problems (e.g. the dining philosophers' problem, leader election and consensus problems). Probabilistic modelling has helped to analyse and reason about the correctness of probabilistic algorithms, to predict system behaviour based on the calculation of performance characteristics and to represent and quantify other forms of uncertainty. The study of probabilistic model checking techniques has been a rich research area.

A great many probabilistic variants of the classical process calculi have also appeared in the literature. The typical approach is to add probabilities to existing models and techniques that have already proved successful in the nonprobabilistic settings. The distinguishing feature of probabilistic process calculi is the presence of a *probabilistic-choice* operator, as in the probabilistic extensions of CCS [6, 7], the probabilistic CSP [8], the probabilistic ACP [9] and the probabilistic asynchronous π-calculus [10].

In order to study a programming language or a process calculus, one needs to assign a consistent meaning to each program or process under consideration. This meaning is the *semantics* of the language or calculus. Semantics is essential to verify or prove that programs behave as intended. Generally speaking, there are three major approaches for giving semantics to a programming language. The *denotational* approach [11] seeks a valuation function that maps a program to its mathematical meaning. This approach has been very successful in modelling many sequential languages; programs are interpreted as functions from the domain of input values to the domain of output values. However, the nature of interaction is much more complex than a mapping from inputs to outputs, and so far the denotational interpretation of concurrent programs has not been as satisfactory as the denotational treatment of sequential programs.

The *axiomatic* approach [12, 13] aims at understanding a language through a few axioms and inference rules that help to reason about the properties of programs. It offers an elegant way of gaining insight into the nature of the operators and the equivalences involved. For example, the difference between two notions of program equivalence may be characterised by a few axioms, particularly if adding these axioms to a complete system for one equivalence gives a complete system for the other equivalence. However, it is often difficult and even impossible to achieve a fully complete axiomatic semantics if the language in question is beyond a certain expressiveness.

The *operational* approach has been shown to be very useful for giving semantics of concurrent systems. The behaviour of a process is specified by its *structural operational semantics* [14], described via a set of labelled transition rules inductively defined on the structure of a term. In this way each process corresponds to a labelled *transition graph*. The shortcoming of operational semantics is that it is too concrete, because a transition graph may contain many states that intuitively should be identified. Thus, a great number of equivalences have been proposed, and different transition graphs are compared modulo some equivalence relations. Usually

there is no agreement on which is the best equivalence relation; in formal verification different equivalences might be suitable for different applications. Sometimes an equivalence is induced by a preorder relation, by taking the intersection of the preorder with its inverse relation, instead of being directly defined.

Among the various equivalences, *bisimilarity* [2, 15] is one of the most important ones as it admits beautiful characterisations in terms of fixed points, modal logics, coalgebras, pseudometrics, games, decision algorithms, etc. In this book we will characterise bisimilarity for probabilistic processes from metric, logical and algorithmic perspectives.

Preorders can be used to formalise a "better than" relation between programs or processes, one that has its origins in the original work assigning meanings to programs and associating a logic with those meanings [12, 13]. Usually that relation is expressed in two different ways: either to provide a witness for the relation or to provide a testing context to make obvious that one program is actually not better than another.

Two important kinds of preorders are *testing preorders* [16, 17] and *simulation preorders*. They give rise to *testing semantics* and *simulation semantics*, respectively. In a testing semantics, two processes can be compared by experimenting with a class of tests. Process P is deemed "better" than process Q if the former passes every test that the latter can pass. In contrast, to show that P is not "better" than Q it suffices to find a test that Q can pass but P cannot. In a simulation semantics, process P can simulate Q if Q performs an action and evolves into Q' then P is able to exhibit the same action and evolve into P' such that P' can simulate Q' in the next round of the simulation game. Simulation is coinductively defined and comes along with a proof principle called *coinduction*: to show that two processes are related it suffices to exhibit a simulation relation containing a pair consisting of the two processes. In the nonprobabilistic setting, simulation semantics is in general finer than testing semantics in that it can distinguish more processes. However, in this book we will see that for a large class of probabilistic processes, the gap between simulation and testing semantics disappears. Therefore, in this case we have a semantics where both negative and positive results can easily be proved: to show that two processes are not related in the semantics we just give a witness test, while to show that two processes are related we construct a relation and argue that it is a simulation relation.

1.2 Synopsis

The remainder of the book is organised as follows. Chapter 2 collects some fundamental concepts and theorems in a few mathematical subjects such as lattice theory, topology and linear programming. They are briefly reviewed and meant to be used as references for later chapters. Most of the theorems are classic results, and thus are stated without proofs as they can be easily found in many standard textbooks in mathematics. It is not necessary to go through the whole chapter; readers can refer to relevant parts of this chapter when it is mentioned elsewhere in the book.

Chapter 3 introduces an operational model of probabilistic systems called *probabilistic labelled transition systems*. In this model, a state might make a nondeterministic choice among a set of available actions. Once an action is taken, the state evolves into a distribution over successor states. Then in order to compare the behaviour of two states, we need to know how to compare two distributions. There is a nice lifting operation that turns a relation between states into a relation between distributions. This operation is closely related to *the Kantorovich metric* in mathematics and the *network flow* problem in optimisation theory. We give an elementary account of the lifting operation because it entails a neat notion of probabilistic bisimulation that can be characterised by behavioural pseudometrics and decided by polynomial algorithms over finitary systems. We also provide modal characterisations of the probabilistic bisimulation in terms of probabilistic extensions of *the Hennessy–Milner logic* and the *modal mu-calculus*.

Starting from Chap. 4 we investigate the testing semantics of probabilistic processes. We first set up a general testing framework that can be instantiated into a vector-based testing or scalar testing approach, depending on the number of actions used to indicate success states. A fundamental theorem is that for finitary systems the two approaches are equally powerful. In order to prove this result we make use of a notion of reward testing as a stepping stone. The *separation hyperplane theorem* from discrete geometry plays an important role in the proof.

Chapter 5 investigates the connection between testing and simulation semantics. For *finite processes*, i.e. processes that correspond to probabilistic labelled transition systems with finite tree structures, testing semantics is not only sound but also complete for simulation semantics. More specifically, may testing preorder coincides with simulation preorder and must testing preorder coincides with failure simulation preorder. Therefore, unlike the traditional (nonprobabilistic) setting, here there is no gap between testing and simulation semantics. To prove this result we make use of logical characterisations of testing preorders. For example, each state s has a characteristic formula ϕ_s in the sense that another state t can simulate s if and only if t satisfies ϕ_s. We can then turn this formula ϕ_s into a characteristic test T_s so that if t is not related to s via the may testing preorder then T_s is a witness test that distinguishes t from s. Similarly for the case of failure simulation and must testing. We also give a complete axiom system for the testing preorders in the finite fragment of a probabilistic CSP. This chapter paves the way for the next chapter.

In Chap. 6 we extend the results in the last chapter from finite processes to *finitary processes*, i.e. processes that correspond to probabilistic labelled transition systems that are *finite-state* and *finitely branching* possibly with loops. The soundness and completeness proofs inherit the general schemata from the last chapter. However, the technicalities are much more subtle and more interesting. For example, we make a significant use of subdistributions. A key topological property is that from any given subdistribution, the set of subdistributions reachable from it by weak transitions can be finitely generated. The proof is highly nontrivial and involves techniques from *Markov decision processes* such as rewards and static policies. This result enables us to approximate coinductively defined relations by stratified inductive relations. As a

consequence, if two processes behave differently we can tell them apart by a finite test.

We also introduce a notion of real-reward testing that allows for negative rewards. It turns out that real-reward may preorder is the inverse of real-reward must preorder, and vice versa. More interestingly, for finitary convergent processes, real-reward must testing preorder coincides with nonnegative-reward testing preorder.

In Chap. 7 we introduce a notion of weak probabilistic bisimulation simply by taking the symmetric form of the simulation preorder given in Chap. 6. It provides a sound and complete proof methodology for an extensional behavioural equivalence, a probabilistic variant of the traditional *reduction barbed congruence* well known in concurrency theory.

References

[1] Barendregt, H.: The Lambda Calculus: Its Syntax and Semantics. North-Holland, Amsterdam (1984)
[2] Milner, R.: Communication and Concurrency. Prentice Hall, Englewood Cliffs (1989)
[3] Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Englewood Cliffs (1985)
[4] Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Inf. Comput. 60, 109–137 (1984)
[5] Baeten, J.C.M., Weijland, W.P.: Process Algebra, Cambridge Tracts in Theoretical Computer Science, vol. 18. Cambridge University Press, Cambridge (1990)
[6] Giacalone, A., Jou, C.C., Smolka, S.A.: Algebraic reasoning for probabilistic concurrent systems. Proceedings of IFIP TC2 Working Conference on Programming Concepts and Methods (1990)
[7] Hansson, H., Jonsson, B.: A calculus for communicating systems with time and probabilities. Proceedings of IEEE Real-Time Systems Symposium. IEEE Computer Society Press, 278–287 (1990)
[8] Lowe, G.: Probabilities and priorities in timed CSP. Ph.D. Thesis, Oxford (1991)
[9] Andova, S.: Process algebra with probabilistic choice. Tech. Rep. CSR 99-12, Eindhoven University of Technology (1999)
[10] Herescu, O.M., Palamidessi, C.: Probabilistic asynchronous pi-calculus. Tech. Rep., INRIA Futurs and LIX (2004)
[11] Scott, D., Strachey, C.: Toward a mathematical semantics for computer languages. Technical Monograph PRG-6, Oxford University Computing Laboratory (1971)
[12] Floyd, R.W.: Assigning meanings to programs. Proc. Am. Math. Soc. Symp. Appl. Math. 19, 19–31 (1967)
[13] Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM 12(10), 576–580 (1969)
[14] Plotkin, G.: The origins of structural operational semantics. J Log Algebraic Program 60–61, 3–15 (2004)
[15] Park, D.: Concurrency and automata on infinite sequences. Proceedings of the 5th GI-Conference on Theoretical Computer Science, Lecture Notes in Computer Science, vol. 104, Springer, 167–183 (1981)
[16] De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theor. Comput. Sci. 34, 83–133 (1984)
[17] Hennessy, M.: Algebraic Theory of Processes. The MIT Press, Cambridge (1988)