



The United Nations  
University

**UNU-IIST**

International Institute for  
Software Technology

---

# Unifying Probability with Nondeterminism

---

**Yifeng Chen and J. W. Sanders**

October 2008

## UNU-IIST and UNU-IIST Reports

UNU-IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macao, and was founded in 1991. It started operations in July 1992. UNU-IIST is jointly funded by the government of Macao and the governments of the People's Republic of China and Portugal through a contribution to the UNU Endowment Fund. As well as providing two-thirds of the endowment fund, the Macao authorities also supply UNU-IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU-IIST is to assist developing countries in the application and development of software technology.

UNU-IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU-IIST, and
7. Dissemination, in which UNU-IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU-IIST is on formal methods for software development. UNU-IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU-IIST produces a report series. Reports are either Research  R, Technical  T, Compendia  C or Administrative  A. They are records of UNU-IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU-IIST at P.O. Box 3058, Macao or visit UNU-IIST's home page: <http://www.iist.unu.edu>, if you would like to know more about UNU-IIST and its report series.

Govindan Parayil, Director, a.i.



The United Nations  
University

**UNU-IIST**

International Institute for  
Software Technology

P.O. Box 3058

Macao

---

# Unifying Probability with Nondeterminism

---

**Yifeng Chen and J. W. Sanders**

## **Abstract**

Any model for sequential programming that contains both (demonic) nondeterminism and probabilism must reveal the difference between these two well-known programs: the first assigns to some variable a nondeterministic choice between Booleans and then assigns to a different variable a fair choice between Booleans, whilst the second reverses the order by making the fair choice before the nondeterministic one. The interest and subtlety of those programs lies in the fact that without probability, disjoint assignments commute and so those programs might naively have been expected to be indistinguishable. Previous relational models separate them by exploiting a complex definition of sequential composition. This paper provides an alternative approach in which sequential composition is instead standard relational composition. The technical contribution that enables that to be achieved expands the binary (initial-final) state view of computation to incorporate a third state, the ‘original’ state which checkpoints the most recent nondeterministic choice. That enables a nondeterministic choice to be made on the basis of past probabilistic choices and so facilitates independent nondeterministic combinations to be chosen against them. The resulting model has the standard relational model embedded in it by a Galois connection.

**Yifeng Chen** has recently moved to Peking University, China, after having been a Senior Lecturer at the University of Durham, England. His interests range over Formal Methods, Compilation, Type Theory and Programming. He is a Research Associate of UNU-IIST.

**Jeff Sanders** is Principal Research Fellow at UNU-IIST. His interests lie largely in Formal Methods.

This work was supported by the Macao Science and Technology Development Fund, under the PEARL project, grant number 041/2007/A3.

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>A relational model for probabilism</b>	<b>3</b>
2.1	Conditional distributions . . . . .	3
2.2	The semantic model . . . . .	5
2.3	Healthiness conditions . . . . .	6
2.4	Algebraic laws and normal form . . . . .	7
2.5	Galois connection with the non-probabilistic model . . . . .	9
<b>3</b>	<b>Examples</b>	<b>11</b>
3.1	Motivating example . . . . .	11
3.2	Case study: varying Monty Hall . . . . .	13
<b>4</b>	<b>Related and further work</b>	<b>15</b>
<b>5</b>	<b>Conclusion</b>	<b>17</b>



## 1 Introduction

It seems inevitable that as more practical but challenging features are added to programming and development languages, the semantic models become more intricate and so less useful. The only way around that sobering difficulty is to structure the semantic models in some way so that their complexity is at least revealed incrementally, feature by feature.

One successful method for adding complexity incrementally has been suggested by Hoare and He in the ‘Unifying Theories of Programming’ (UTP) approach [7, 4] to program semantics. By relating increments via Galois connections, the method is ‘economical’, ensuring that previous laws are inherited at each new increment. Being founded on a relational model of initial/final states, the method is supported by a powerful underlying ‘operational intuition’ and can take advantage of the substantial theory of relations. For example, a sequential composition ( $A \ ; \ B$ ) should relate the final state of  $A$  and the initial state of  $B$ ; the initial state of  $A$  is not directly related to any state of  $B$ , reflecting the sequential interaction between two components. Another example is the modelling of nondeterministic choice by union.

Yet the UTP approach must be doomed to fail in any situation where the features being introduced incrementally are too tightly coupled, and so cannot be teased out *seriatum*. Fortunately it has already proved successful in accounting for enabledness (as observable  $ok$ ), termination (as observable  $ok'$ ), abstraction (*i.e.* demonic nondeterminism) (as disjunction), sequential programming with recursion (and compilation), reactivity and various forms of concurrency (with interaction by merge)—perhaps the most stringent test to date—and functional programming. Partial success has been obtained with object orientation and mutable data structures; but applying the UTP approach to probability has proved more challenging [6].

In standard probability theory, two random events are either dependent or independent. Repeatedly throwing two (fair) coins yields equal probability for each of them to land heads-up, and equal probability for them to show the same face (both heads or both tails). However this kind of independency is not the least known relationship between two events. In software development, there are factors unknown or chosen to be ignored at a certain level of abstraction. This is called *nondeterminism*. The least known relation between two events is actually that one event is unknown to depend on the other. In the worst case, we must assume that the later event occurs according to the probabilistic choice of an earlier event and violates certain desirable properties. The distinction between probability and nondeterminism is particularly necessary in the design of a security system where the adversary is nondeterministic without the assumption of independency.

Without abstraction—that is, without demonic nondeterminism—there is no difficulty unifying probabilism with sequential programming. But realistically we are interested only in formalisms sufficiently powerful to be used in practice. That means they must apply across a substantial part, and preferably all, of the hierarchy of development from specification to implementation. They must therefore include abstraction reflecting local blocks and the requirement of local reasoning which arises unavoidably in Software Engineering. Furthermore the formalism must reflect the absence of a known relationship between events. Thus it must contain nondeterminism.

We consider probabilism, as has become standard, in the guise of a binary combinator  $A_p \oplus B$  that chooses program  $A$  with probability  $p$  (a  $[0,1]$ -valued expression on state space that sums to at most 1) and  $B$  with probability  $1 - p$ . That suffices to express known probabilistic algorithms and to reason about a moderately broad range of probabilistic behaviour [8]. It is shown there that Dijkstra's guarded-command language augmented with  $_p \oplus$  provides a remarkably simple uniform notation facilitating the simultaneous treatment of functional *and* probabilistic behaviour. So with almost no extra complexity, probabilistic behaviour need not be handled by a 'second pass', but all at once with input/output behaviour.

The expectation must then be that probabilism can be added to the guarded-command language via the UTP approach. One of the reasons that has not been done is encapsulated in this example, demonstrating the complex interaction between nondeterminism and probabilism

$$(1) \quad (x := 0 \sqcap x := 1) \wp (y := 0 \frac{1}{2} \oplus y := 1) \neq (y := 0 \frac{1}{2} \oplus y := 1) \wp (x := 0 \sqcap x := 1)$$

where  $x := 0$  represents assignment,  $\frac{1}{2} \oplus$  chooses either side with equal probability, and  $\sqcap$  represents nondeterministic choice. Indeed, the probabilistic choice on the left-hand side is made *after* a value is assigned to  $x$  by the nondeterministic choice. Thus the probability of guaranteeing the condition  $x = y$  is  $\frac{1}{2}$ . But the right-hand program behaves more nondeterministically: the nondeterministic choice can observe, and so exploit, the preceding probabilistic choice, so that in the worst case it may keep ensuring  $x \neq y$ ; thus the probability of guaranteeing  $x = y$  is reduced to 0.

Thus, in the presence of both probability and nondeterminism, disjoint assignments no longer commute! Of course in standard sequential programming, disjoint assignments commute as a result of the two distributivity laws:

$$\begin{aligned} (B \sqcap C) \wp A &= (B \wp A) \sqcap (C \wp A) \\ A \wp (B \sqcap C) &= (A \wp B) \sqcap (A \wp C). \end{aligned}$$

It is the second which fails in probabilistic programming because the nondeterministic choice after  $A$  can observe, and so exploit, more probabilistic history than the choice before  $A$ . Thus only this holds

$$(2) \quad A \wp (B \sqcap C) \sqsubseteq (A \wp B) \sqcap (A \wp C).$$

By comparison, in the first law the nondeterministic choice is made initially on both sides and so there is no scope for any difference.

Two models of probability and nondeterminism have been developed in which the revised laws are sound: an expectation-transformer model [10] corresponding to the predicate-transformer model of standard programming, and a distributional model [5], corresponding to the standard binary-relational model. In the transformer model, each computation is modelled by transforming a random variable over final states (seen as an expectation) to the (pointwise) greatest expected value over initial states that can be guaranteed by executing the computation with 'reward' the post random variable. In the distributional

model, each computation is seen as transforming an initial state to a set of distributions over final states (though in this paper we prefer to make it a relation between distributions). The distributional model is embedded in the transformer model by a Galois connection [8].

In the transformer model, Law (2) follows by monotonicity of expectation transformers (a simple consequence of their characteristic healthiness condition ‘sublinearity’). But in the distributional model where nondeterminism is union, if sequential composition were to be the usual sequential composition of relations then equality would hold in (2). Thus the definition of sequential composition in [5] is more sophisticated.

Here we follow the alternative of using the standard relational definition of sequential composition but a slightly more sophisticated definition of nondeterminism. We start from the observation that nondeterministic choices made after a sequential composition may be correlated with what happened well before. Thus we introduce a model of computation that enables a ‘demonic log’ to be maintained and a ‘demonic checkpoint’ to be taken to reset the log. The state from which the log extends is called the ‘original’ state of the computation; and we consider distributions over final state. A nondeterministic choice is resolved by reference to the log, which in particular reveals all probabilistic choice made since the original state. But when the log is reset, original and current states coincide and there is no history for the demon to exploit; the result is ‘nondeterministically closed’.

The model and its basic theory are considered in the next section. In Section 3 the motivating example and the familiar Monty-Hall problem are analysed, and the latter varied to demonstrate the strength of the new model. Related and further work are given in Section 4 and conclusions drawn in Section 5.

## 2 A relational model for probabilism

We begin by introducing the notion of ‘original state’ and the relevant definition of ‘conditional probability distribution’. The semantic model consists of seven basic commands from which further useful commands are derived. Several healthiness conditions are introduced. Algebraic laws are identified and used to transform programs into a normal form. A Galois connection is established from the standard relational model into this model.

### 2.1 Conditional distributions

Let  $S \hat{=} V \rightarrow C$  be the set of all states, each a mapping from program variables to constants. A (conditional probability) *distribution* is a function:  $h : S \rightarrow (S \rightarrow [0, 1])$  whose first argument  $s$  denotes the *original state*, and second argument denotes the *current state*. For example,  $\lambda s, t : S \cdot 1$  is a constant distribution for all states.

The conjunction  $h_1 \& h_2$  between two distributions is defined to be the pairwise minimum:

$$(h_1 \& h_2).s.t = \min(h_1.s.t, h_2.s.t)$$

which definition bears obvious similarity with conjunction from fuzzy logic. We adopt the convention that Booleans are written numerically, *true* is 1 and *false* is 0, and thence that

$$\lceil S_0 \rceil.s.t \hat{=} (s \in S_0).$$

For example,  $\lceil x=y \rceil$  denotes  $\lceil S_0 \rceil$  where  $S_0 = \{s \in S \mid s(x)=s(y)\}$ . We also adopt the convention:  $\lfloor S_0 \rfloor.s.t \hat{=} (t \in S_0)$ .

A distribution  $d$  is defined to be *well-formed* iff from any original state  $s$ , the sum probability of all current states is at most 1:

$$\sum_t d.s.t \leq 1.$$

The original state refers to some state before (or equal to) the current state. A well-formed distribution is a probability distribution over (current) states conditioned on a specific original state. A well-formed distribution satisfying  $d.s.t=p$  records that if the *original state* is  $s$ , then the probability of *reaching this point* of the program in a *current state*  $t$  is  $p$ . The explanation implies that a nonterminating program results in a total probability of less than 1. Let  $\delta$  denote a special well-formed distribution defined  $\delta.s.t \hat{=} (s=t)$ .

The *inner product* (over the current states from every original state) between a well-formed distribution and a distribution is defined:

$$(d \cdot h).s \hat{=} \sum_t d.s.t * h.s.t.$$

The result with signature  $S \rightarrow [0, 1]$  is a mapping from original states to some ‘choosing factors’ (think of the probabilistic expression in a probabilistic choice). Such a mapping is called a *factor function*.

The probabilistic combination between probabilities is defined:  $p_1 \oplus p_2 \hat{=} p \times p_1 + (1-p) \times p_2$ . The lifted probabilistic combination between well-formed distributions is defined:

$$(d_1 \oplus_e d_2).s.t \hat{=} d_1.s.t \oplus_{e.s} d_2.s.t$$

where  $e$  is a factor function. This definition allows the choosing factors (probabilities) to depend on the original states.

The *functional update* of a well-formed distribution is defined:

$$(d \dagger f).s.t \hat{=} \sum_{u: f(s,u)=t} d.s.u$$

where  $f: S \rightarrow (S \rightarrow S)$  is a state function. From any original state (*i.e.* the first argument), the probability of a final state is the sum of all probabilities of the initial states mapped into the final state.

The (current-state) composition  $f \circ g$  of two state functions is defined to be  $(f \circ g).s.t = f.s.(g.s.t)$ . Note that the same notation is also applicable when  $f$  is a distribution.

The *involution composition* between two distributions is defined:

$$(d_1 \otimes d_2).s.t \hat{=} \sum_u d_1.s.u * d_2.u.t.$$

This composition represents the conditional probabilities dependent on the original states. Involution is linear and hence distributes over convex combinations:

$$\begin{aligned} d \otimes (d_1 \oplus d_2) &= (d \otimes d_1) \oplus (d \otimes d_2) \\ (d_1 \oplus d_2) \otimes d &= (d_1 \otimes d) \oplus (d_2 \otimes d) \end{aligned}$$

A Boolean expression  $b: S \rightarrow (S \rightarrow \{0, 1\})$  is a special well-formed distribution. For example, the Boolean expression  $s=t$  is the same as  $\delta$ . The expression  $(x=y)$  is shorthand for the function  $b.s.t=1$  when  $t(x)=t(y)$ , and  $b.s.t=0$  otherwise.

We shall use  $s$  to denote the original state,  $t$  to denote the current state,  $h, h_1, \dots$  to denote arbitrary distributions,  $d, d_1, \dots$  to denote well-formed distributions,  $b, b_1, \dots$  to denote Boolean expressions,  $e, e_1, \dots$  to denote factor functions,  $f, g, f_1, \dots$  to denote state functions, and  $x, y, x_1, \dots$  to denote program variables (and their values in the current state).

## 2.2 The semantic model

A computation is represented by a relation  $A(d, d')$  between an initial (well-formed) distribution  $d$  and a final (well-formed) distribution  $d'$ , reflecting the probability distributions of the initial and final states from specific original states.

---


$$\begin{aligned} \perp &\hat{=} true \\ A \ ; \ B &\hat{=} \exists d_0 \cdot A(d, d_0) \wedge B(d_0, d') \\ A \ ;_h \ B &\hat{=} \exists d_1 d_2 \cdot A(d, d_1) \wedge B(d, d_2) \wedge d' = d_1 \ ;_d \ ;_h \oplus d_2 \\ A \oplus B &\hat{=} \exists h \cdot A \ ;_h \ B \\ \square(A) &\hat{=} \exists d_0 \cdot A(\delta, d_0) \wedge d' = d \otimes d_0 \\ t := f &\hat{=} d' = d \ ;_t \ f \ ;_1 \oplus \perp \\ \mu F &\hat{=} \bigvee \{A \mid A \ ;_1 \ F(A)\} \end{aligned}$$


---

Abort  $\perp$ , representing nontermination, is the most nondeterministic computation and may end up in any distribution from every distribution. Sequential composition  $A \ ; \ B$  relates, and hides, the final distribution of  $A$  and the initial distribution of  $B$ . Probabilistic choice  $A \ ;_h \ B$  linearly combines the result distributions from  $A$  and  $B$  where  $h$  is a distribution for the weights over initial states.

*Pure nondeterminism*  $A \oplus B$  applies arbitrary convex combinations between distributions depending on both original and current states. From different original states, the resulting distributions may be linearly combined with different factors. Nondeterministic closure  $\square(A)$  resets the initial state of  $A$  to become

the original state; it is achieved with the delta distribution  $\delta$ . The results from all possible initial states are recorded in  $d_0$ , representing how the final distributions depend on the initial states. The result distribution is multiplied with the initial distribution  $d$  to reflect the influence of the probability distribution of initial states.

The definition of closure illustrates how a computation can take advantage of the history (as far back as the beginning of the closure). The assignment  $t := f(s, t)$  uses a total function  $f$  to modify the state according to the initial state  $t$  and the original state  $s$ . The probability of a final state is the sum of probabilities of the initial states mapped into the final state, all dependent on the same original states. The full binary conditional with abort requires that if a computation may not terminate, the distribution of nontermination is chaotic. Recursion is defined as the weakest fixpoint.

Useful derived commands are as follows.

$\Pi$	$\hat{=} t := t$
$\tilde{\Pi}$	$\hat{=} t := s$
$A \sqcap B$	$\hat{=} \square(A \oplus B)$
$A \triangleleft b \triangleright B$	$\hat{=} A_b \oplus B$
$A \ominus B$	$\hat{=} A \triangleleft \delta \triangleright B$

Skip  $\Pi$  maintains the current state. Compensator  $\tilde{\Pi}$  performs the compensation and restores the current state to the original state. Backward-looking nondeterministic choice  $A \sqcap B$  corresponds to that of the standard probabilistic models and performs arbitrary convex combinations within a nondeterministic closure. The convex combinations are relative to the state at the beginning of the closure. Section 3.1 will show how this enables the nondeterminism to act against the original probabilistic choices.

Binary conditional  $A \triangleleft b \triangleright B$  is a special probabilistic choice. Note that Boolean expressions are also distributions. The state-change choice  $A \ominus B$  chooses  $A$  with the probability for the current state being equal to the original state. This operator can be used to detect state change and perform more sophisticated compensation.

Note that finite disjunction is not closed in the semantics, but that arbitrary disjunction (of probabilistic choices) is.

### 2.3 Healthiness conditions

Healthiness conditions record assumed desirable properties of a language. The total probability (of initial states) from each original state is at most 1. When it is less than 1, the deficiency represents the probability of nontermination. A principle of the UTP approach [7] and other totally-correct models is to assume that if the computation has not started then the computation becomes chaotic. In our probabilistic model, this is ensured by a healthiness condition (corresponding to H1 of UTP [7]):

$$A = (A \triangleleft 1 \triangleright \perp).$$

Note that the (constant) choice function  $\perp$  does not guarantee to choose  $A$  on the left. Instead, it chooses  $\perp$  with non-zero probability when the initial distribution  $d$  does not always terminate (with total probability less than 1 from a given original state).

Another healthiness condition ensures that if the computation may not terminate, the final distribution is chaotic for the probability of nontermination (corresponding to H3 of UTP):

$$A = (A \circledast \perp).$$

Note that  $\perp$  itself also satisfies the first healthiness condition, which ensures a chaotic final distribution on nontermination.

Arbitrary convex combinations of the final distributions (independently) from each initial state are closed (*i.e.*  $\oplus$  is idempotent):

$$A = (A \oplus A).$$

That healthiness condition implies the idempotence of probabilistic choice in Law 2(1).

In this paper, we assume that all computations are feasible (*i.e.* free of miracles) so that from any initial distribution, there exists some final distribution (corresponding to H4 of UTP):

$$(A \circledast \perp) = \perp.$$

The fixpoint of our model uses Tarski's fixpoint theory and hence does not require the healthiness condition of Cauchy closure for continuity.

A specification  $A$  is called *nondeterministically closed* if  $A = \square(A)$ . Such a computation does not depend on the original state. An assignment  $t := f$  is closed if  $f$  does not depend on the original state  $s$ . A probabilistic choice  $A \oplus B$  is closed if  $h$  does not depend on the original state. The choice  $A \oplus B$  is open unless the arguments are special. The sequential composition  $A \circledast B$  is closed if both  $A$  and  $B$  are closed. Standard probabilistic programming corresponds to the sub theory of nondeterministically closed specifications in the new model.

## 2.4 Algebraic laws and normal form

The algebraic laws of this section are semantically sound. Firstly we now identify laws of the basic commands. Note that the second halves of Law 1(1) and (2) are actually assumed as healthiness conditions in the previous subsection.

- Law 1**
- (1)  $\perp \circledast A = \perp = A \circledast \perp$
  - (2)  $\perp \circledast A = A = A \circledast \perp$
  - (3)  $t := f \circledast t := g = t := g \circ f$
  - (4)  $(A \circledast B) \circledast C = A \circledast (B \circledast C)$

The following laws identify various properties of probabilistic choice.

- Law 2**
- (1)  $A \text{ }_h\oplus A = A$
  - (2)  $A \text{ }_h\oplus B = B \text{ }_{1-h}\oplus A$
  - (3)  $(A \text{ }_h\oplus B) \text{ }_{h'}\oplus C = A \text{ }_{hh'}\oplus (B \text{ }_{h''}\oplus C) \quad (h''(1-hh')=h'(1-h)).$
  - (4)  $(A \text{ }_h\oplus B) \text{ } ; C = (A \text{ } ; C) \text{ }_h\oplus (B \text{ } ; C)$
  - (5)  $t := f \text{ } ; (A \text{ }_h\oplus B) = (t := f \text{ } ; A) \text{ }_{h\circ f}\oplus (t := f \text{ } ; B)$

Pure nondeterministic choice  $\oplus$  is similar to nondeterminism in the (non-probabilistic) sequential model. In particular, it satisfies right distributivity for sequential composition, which does not hold in previous probabilistic models, suggesting that the operator does not exploit history.

- Law 3**
- (1) *The operator  $\oplus$  is idempotent, commutative and associative.*
  - (2) *Sequential composition distributes over  $\oplus$ .*
  - (3) *Probabilistic choice distributes over  $\oplus$ .*

Nondeterministic closure is strict and idempotent. It forces current state to coincide with the original state for the assignment. The closure distributes over pure nondeterministic choice. If the second half of a sequential composition in a closure is closed, the outer closure can be decomposed into two closures in a sequential composition. The closure also forces a probabilistic choice to equate the original and current states in the choosing factor.

- Law 4**
- (1)  $\square(\perp) = \perp$
  - (2)  $\square(\square(A)) = \square(A)$
  - (3)  $\square(\Pi) = \Pi$
  - (4)  $\square(t := f) = t := f(t, t)$
  - (5)  $\square(A \oplus B) = \square(\square(A) \oplus B)$
  - (6)  $\square(A \text{ } ; \square(B)) = \square(A) \text{ } ; \square(B)$
  - (7)  $\square(A \text{ }_h\oplus B) = \square(A) \text{ }_{h\&\delta}\oplus \square(B)$

The above laws are enough to transform any finite program into the following normal form **N**:

$$\begin{aligned} \mathbf{N} &::= \mathbf{M} \mid \mathbf{N} \oplus \mathbf{N} \\ \mathbf{M} &::= \mathbf{L} \mid \mathbf{M} \text{ }_h\oplus \mathbf{M} \\ \mathbf{L} &::= \perp \mid \Pi \mid t := f \text{ } ; \square(\mathbf{N}). \end{aligned}$$

Alternatively, a program in normal form can be written as follows:

$$A = \bigoplus_i \bigoplus_{j, h_{ij}} t := f_{ij} \text{ } ; \square(A_{ij})$$

where  $A_{ijk}$  can be abort  $\perp$ ,  $\Pi$  or a program in normal form. The following (relative) completeness theorem is proved with the algebraic laws.

**Theorem 1 (Completeness)** *For every program in the model, there exists a semantically equal program in  $\mathbf{N}$ .*

The laws of derived commands are provable from those for the basic commands. Nondeterministic choice  $\sqcap$  is commutative and associative. For example, associativity follows:

$$\begin{aligned} (A \sqcap B) \sqcap C &= \sqcap(\sqcap(A \oplus B) \oplus C) \\ &= \sqcap((A \oplus B) \oplus C) \\ &= \sqcap(A \oplus (B \oplus C)) \\ &= \sqcap(A \oplus \sqcap(B \oplus C)) \\ &= A \sqcap (B \sqcap C). \end{aligned}$$

If a program  $A$  is nondeterministically closed, then  $A \sqcap A = A$ . In general,  $A \sqcap A = \sqcap(A)$ . Remember that all programs in previous models correspond to nondeterministically closed specifications of this model. Sequential composition distributes nondeterministic choice on the left when the right-hand program is nondeterministically closed:

$$\begin{aligned} (A \sqcap B) \ ; \ \sqcap(C) &= \sqcap(A \oplus B) \ ; \ \sqcap(C) \\ &= \sqcap((A \oplus B) \ ; \ \sqcap(C)) \\ &= \sqcap((A \ ; \ \sqcap(C)) \oplus (B \ ; \ \sqcap(C))) \\ &= (A \ ; \ \sqcap(C)) \sqcap (B \ ; \ \sqcap(C)). \end{aligned}$$

As in the previous models, sequential composition does not distribute nondeterminism from the right.

The compensator  $\tilde{\Pi}$  can restore the original state to the current state and compensate for any state change:

$$\sqcap(A \ ; \ \tilde{\Pi} \ ; \ B) = \sqcap(B).$$

More sophisticated compensation can be achieved with the state-change detector  $\ominus$ , which satisfies the following interesting law:

$$\sqcap(A \ominus B) = \sqcap(A).$$

## 2.5 Galois connection with the non-probabilistic model

The standard relational model of sequential programming is based on total correctness. Each program is represented as a relation between (generalised) states  $S_{\uparrow}$  where  $\uparrow$  denotes nontermination and where  $S_{\uparrow} \hat{=} S \cup \{\uparrow\}$ . The program that never terminates is the full relation and denoted  $\bar{\emptyset}$ . Other commands include sequential composition as relational composition  $r_1; r_2$ , nondeterministic choice as set union  $r_1 \cup r_2$ , binary conditional  $r_1 \triangleleft b \triangleright r_2$  where  $b \subseteq S$ , assignment  $t := f(t)$  (a function of current state, naturally) and recursion:

---


$$\begin{aligned}
\overline{\emptyset} &\hat{=} S_{\uparrow} \times S_{\uparrow} \\
r_1; r_2 &\hat{=} \{(t_1, t_3) \mid \exists t_2 \cdot (t_1, t_2) \in r_1 \wedge (t_2, t_3) \in r_2\} \\
r_1 \cup r_2 &\hat{=} r_1 \cup r_2 \\
r_1 \triangleleft b \triangleright r_2 &\hat{=} (r_1 \cap b_{\uparrow} \times S_{\uparrow}) \cup (r_2 \cap \overline{b}_{\uparrow} \times S_{\uparrow}) \\
t := f(t) &\hat{=} \{(t, f(t)) \mid t \in S\} \triangleleft S \triangleright \overline{\emptyset} \\
\mu F &\hat{=} \bigcup \{r \mid r \subseteq F(r)\}.
\end{aligned}$$


---

Relations representing programs satisfy these healthiness conditions:

- chaotic if not enabled:  $r = r \triangleleft S \triangleright \overline{\emptyset}$
- chaotic if not terminating:  $r = r; (t := t)$
- feasible:  $r; \overline{\emptyset} = \overline{\emptyset}$ .

Those conditions directly correspond to the first three healthiness conditions of the probabilistic model, although nondeterministic choice here is, as set union, idempotent by definition:  $r = r \cup r$ .

Each healthy relation in the standard model can be lifted to a computation in our probabilistic model:

$$\sigma(r) \hat{=} \prod_{f: \lambda t. r(t, f(t))} \perp \triangleleft r(t, \uparrow) \triangleright t := f(t).$$

The lifted computation is the nondeterministic choice of all possible deterministic assignments allowed by the relation.

**Proposition 1** *If a relation  $r$  is healthy in the standard model, so is  $\sigma(r)$  in the probabilistic model.*

The mapping forms an injective Galois connection from relational commands to probabilistic commands:

- Law 5**
- (1)  $\sigma(\overline{\emptyset}) = \perp$
  - (2)  $\sigma(r_1; r_2) = \sigma(r_1) \circledast \sigma(r_2)$
  - (3)  $\sigma(\bigcup \mathcal{R}) = \prod \{\sigma(r) \mid r \in \mathcal{R}\}$
  - (4)  $\sigma(r_1 \triangleleft b \triangleright r_2) = \sigma(r_1) \triangleleft b \triangleright \sigma(r_2)$
  - (5)  $\sigma(t := f(t)) = t := f(t)$ .

### 3 Examples

#### 3.1 Motivating example

In this section, we consider Example (1) from the introduction and originally from [5]. For convenience zero initialisation is added to both programs:

$$\begin{aligned} x, y := 0, 0 \ ; \ (x := 0 \sqcap x := 1) \ ; \ (y := 0 \ \frac{1}{2} \oplus y := 1) \\ x, y := 0, 0 \ ; \ (y := 0 \ \frac{1}{2} \oplus y := 1) \ ; \ (x := 0 \sqcap x := 1). \end{aligned}$$

where  $x$  and  $y$  are two Boolean program variables (expressed as either 0 or 1). Thus together they have 4 states, denoted (00) (*i.e.*  $x=y=0$ ), (01), (10) and (11). We introduce the delta distribution  $\delta_{v,u}$  for a given initial state  $u$  from an original state  $v$ :

$$(\delta_{v,u}).s.t \hat{=} ((s,t) = (u,v)).$$

We use question marks to denote the sum of arbitrary matching states. For example the delta distribution from every original state to the current state (00) is denoted:

$$\delta_{?,00} \hat{=} \delta_{00,00} + \delta_{01,00} + \delta_{10,00} + \delta_{11,00}.$$

Question marks appear only on the original state.

We first consider the example with nondeterministic choice preceding probabilistic choice. The initialisation sets the state to be (00) with probability 1 from every original state. The nondeterministic closure resets the distribution with  $\delta$ , equating the current state to the original state. The pure nondeterministic choice performs convex closure with an arbitrary factor distribution  $h$  (as a fresh constant in the program) between the assignments  $x := 0$  and  $x := 1$ .

As only the state (00) has non-zero probability before the closure, the factor function  $\delta \cdot h$  for the resulting distributions simplifies to  $h.00.00$  after the nondeterministic closure. The assignments  $y := 0$  and  $y := 1$  affect only the value of  $y$  in the distribution, while the  $\frac{1}{2} \oplus$ -probabilistic choice combines the result distributions with equal probability. No matter what probability  $h.00.00$  was chosen, the overall probability for  $x=y$  (in the states 00 and 11) is always  $\frac{1}{2}$ .

Using annotations for convenience,

$$\begin{aligned}
& x, y := 0, 0 \\
& \{d = \delta_{??,00}\} \\
& \square \{d = \delta\} \\
& \text{con } h \\
& \quad x := 0 \quad \{d = \delta_{?0,00} + \delta_{?1,01}\} \\
& \quad h \oplus \\
& \quad \quad x := 1 \quad \{d = \delta_{?0,10} + \delta_{?1,11}\} \\
& \quad \quad \{d = (\delta_{?0,00} + \delta_{?1,01}) \delta_{\cdot h} \oplus (\delta_{?0,10} + \delta_{?1,11})\} \\
& \quad \{d = \delta_{??,00} \otimes ((\delta_{?0,00} + \delta_{?1,01}) \delta_{\cdot h} \oplus (\delta_{?0,10} + \delta_{?1,11}))\} \\
& \quad \{d = \delta_{??,00} h.00.00 \oplus \delta_{??,10}\} \\
& \quad \quad y := 0 \quad \{d = \delta_{??,00} h.00.00 \oplus \delta_{??,10}\} \\
& \quad \quad \frac{1}{2} \oplus \\
& \quad \quad \quad y := 1 \quad \{d = \delta_{??,01} h.00.00 \oplus \delta_{??,11}\} \\
& \quad \quad \left\{ d = (\delta_{??,00} h.00.00 \oplus \delta_{??,10}) \frac{1}{2} \oplus (\delta_{??,01} h.00.00 \oplus \delta_{??,11}) \right\} \\
& \quad \left\{ d \cdot [x=y] = h.00.00 \frac{1}{2} \oplus (1-h.00.00) = \frac{1}{2} \right\}
\end{aligned}$$

On the other hand, if probabilistic choice precedes nondeterministic choice, the distribution before the nondeterministic choice will have each state of (00) and (01) with equal probability. That means both (independently chosen)  $h.00.00$  and  $h.01.01$  are exploited for convex combination in the subsequent nondeterminism. The overall probability for  $x=y$  then depends on the choice of  $h$  ranging from 0, when  $h.00.00=0$  and  $h.01.01=1$ , to 1 when  $h.00.00=1$  and  $h.01.01=0$ . So no non-zero probability for  $x=y$  can be *guaranteed*.

$$\begin{aligned}
& x, y := 0, 0 \\
& \{d = \delta_{??,00}\} \\
& \quad y := 0 \quad \{d = \delta_{??,00}\} \\
& \quad \frac{1}{2} \oplus \\
& \quad \quad y := 1 \quad \{d = \delta_{??,01}\} \\
& \quad \left\{ d = \delta_{??,00} \frac{1}{2} \oplus \delta_{??,01} \right\} \\
& \quad \square \{d = \delta\} \\
& \quad \text{con } h \\
& \quad \quad x := 0 \quad \{d = \delta_{?0,00} + \delta_{?1,01}\} \\
& \quad \quad h \oplus \\
& \quad \quad \quad x := 1 \quad \{d = \delta_{?0,10} + \delta_{?1,11}\} \\
& \quad \quad \quad \{d = (\delta_{?0,00} + \delta_{?1,01}) \delta_{\cdot h} \oplus (\delta_{?0,10} + \delta_{?1,11})\} \\
& \quad \left\{ d = (\delta_{??,00} \frac{1}{2} \oplus \delta_{??,01}) \otimes ((\delta_{?0,00} + \delta_{?1,01}) \delta_{\cdot h} \oplus (\delta_{?0,10} + \delta_{?1,11})) \right\} \\
& \quad \left\{ d = (\delta_{??,00} h.00.00 \oplus \delta_{??,10}) \frac{1}{2} \oplus (\delta_{??,01} h.01.01 \oplus \delta_{??,11}) \right\} \\
& \quad \left\{ d \cdot [x=y] = (h.00.00 \frac{1}{2} \oplus (1-h.01.01)) \right\}
\end{aligned}$$

The difference between the two seemingly equal programs is due, semantically, to the fact that pure nondeterministic choice can make different probabilistic choices from different original states. All commands normally share the same original state, but a nondeterministic closure can alter this and reset the original state, allowing the computation within the closure to act against the probabilistic choice in the original state.

### 3.2 Case study: varying Monty Hall

The famous Monty-Hall puzzle (in this context, from [10]) describes a game show with a host, a guest and three closed doors: one of them hides a car, which the player wishes to win, whilst the other two hide goats, which the player intends to avoid. The player begins by choosing a door, but it is not opened immediately. Instead, the host opens a door different from the one just chosen by the guest and then offers the guest the option of switching choice to the other closed door. Contrary to common perception that the chance of winning the car is unchanged by whether or not the guest switches, the correct move is to switch, and it doubles the overall chance from 1/3 (independent initial right choice among three) to 2/3 (initial wrong choice to be corrected with the host's assistance).

The modelling of this puzzle and its solution involve both probability and nondeterminism. Let the variable  $x$  denote the host's initial choice (in program  $HC$ ) of the door (number 1, 2 or 3) for the car, which is completely unknown to the player:

$$HC \hat{=} x := 1 \sqcap (x := 2 \sqcap x := 3).$$

The variable  $y$  denotes the player's choice. The player, with no knowledge of the position of the car, chooses fairly among the three:

$$PC \hat{=} y := 1 \frac{1}{3} \oplus (y := 2 \frac{1}{2} \oplus y := 3).$$

The player's only knowledge is that the host chooses the door before the player's initial choice, and that the host is not prescient. The host's subsequent door opening (denoted with variable  $z$ ) depends on the player's initial choice. If the choice is right, then the host chooses one of the two remaining goat doors; otherwise, the host chooses the only remaining goat door:

$$HC_1 \hat{=} (z := goat_1(x) \sqcap z := goat_2(x)) \triangleleft [x=y] \triangleright z := goat(x,y)$$

where the function  $goat_1(x)$  returns the smaller door number other than  $x$ ,  $goat_2(x)$  returns the larger number, and  $goat(x,y)$ , defined only when  $x \neq y$ , returns the only other number. For example,  $goat_2(2) = 3$  and  $goat(1,2) = 3$ . The player's second choice results either in stay  $ST \hat{=} \Pi$  or switch  $SW \hat{=} y := goat(y,z)$ .

The game corresponds the computation:

$$HC \wp PC \wp HC_1 \wp ?.$$

A routine calculation guarantees probability 1/3 if the question mark is replaced by  $ST$  but 2/3 if that is replaced by  $SW$ . Note that the position of the car must not depend on the player's initial choice. If the host places the car *after* the player's choice:

$$PC \wp HC \wp HC_1 \wp ?.$$

It is *unknown* whether the (nondeterministic) host places the car against or for the player's interests, or indeed chooses neutrally. As a result, no matter what the player does in the end, the strategy cannot even guarantee a small probability of success. That phenomenon is modelled correctly by both this and previous models.

Now consider a less honest host who, though setting the car before the player, has detected the player's tendency to switch. He secretly tries to move the car to the player's first door with probability  $1/3$  (no move if the player is right):

$$HC_2 \hat{=} (x := y \frac{1}{3} \oplus \Pi).$$

The host performs this dishonest act after opening a door but before the player's final choice:

$$HC \wp PC \wp HC_1 \wp HC_2 \wp ?.$$

Now the probability of initial correctness and staying with the original choice is increased to  $1/3 + (2/3 \times 1/3) = 5/9$ , while the probability of success for switching drops to  $4/9$ .

However, an alert player decides to stay whenever he detects any noise of the car's move (*i.e.* detecting a change of state; we assume that the host is not devious enough to move the car around behind the same door) but switch otherwise:

$$APC \hat{=} SW \ominus ST.$$

The player performs this kind of scrutiny by comparing the *original state*, immediately after the host opens a door, with the *current state* before the final decision:

$$HC \wp PC \wp HC_1 \wp \square(HC_2 \wp APC).$$

The car moves with probability  $2/3 \times 1/3 = 2/9$  when the player's initial choice was wrong. It is always favourable for the player to stay after detecting noise. Switching yields probability  $2/3 \times 2/3 = 4/9$  of success for the player's unswitched initial wrong choice (greater than the success probability  $1/3$  for staying). The overall probability of success of *APC* is  $2/9 + 4/9 = 2/3$ . We use three numbers to denote a state. For example, *h.111.222* denotes the value of *h* from original state 111 and current state 222.

A sketch of the reasoning for the last program is:

$$\begin{array}{l}
\Box \Box \{d = \delta\} \qquad \text{host's choice} \\
\quad \mathbf{con} \ h, h' \\
\quad \quad x := 1 \ \frac{1}{2} \oplus (x := 2 \ \frac{1}{2} \oplus x := 3) \ ; \\
\quad \quad \{d = \delta_{???,111} \ \frac{1}{2} \oplus (\delta_{???,211} \ \frac{1}{2} \oplus \delta_{???,311})\} \\
\quad \quad y := 1 \ \frac{1}{3} \oplus (y := 2 \ \frac{1}{2} \oplus y := 3) \ ; \qquad \text{player's choice} \\
\quad \quad \{d \cdot [x=y] = \frac{1}{3}\} \\
\quad \quad (z := \mathit{goat}_1(x) \sqcap z := \mathit{goat}_2(x)) \triangleleft [x=y] \triangleright z := \mathit{goat}(x, y) \ ; \quad \text{host opens a door} \\
\quad \quad \{d \cdot [x=y] = \frac{1}{3} \wedge d \cdot [y \neq x = \mathit{goat}(y, z)] = \frac{2}{3}\} \\
\quad \quad \Box \{d = \delta\} \\
\quad \quad \quad x := y \ \frac{1}{3} \oplus \Pi \ ; \qquad \text{car move} \\
\quad \quad \quad \left\{ d = (f_{x:=y} \circ \delta) \ \frac{1}{3} \oplus \delta \right\} \\
\quad \quad \quad y := \mathit{goat}(y, z) \ \delta \oplus \Pi \qquad \text{alert player's choice} \\
\quad \quad \{d \cdot [x=y] = \frac{2}{3} \times \frac{1}{3} + \frac{2}{3} \times \frac{2}{3} = \frac{2}{3}\}.
\end{array}$$

This case study illustrates how the player and the nondeterministic host can both benefit from the extra information recorded in the original state. The last variation is not captured by previous models.

## 4 Related and further work

It will have been apparent already that this work is most closely related to the distributional model of He *et al.* [5] which appeared more than two decades after Rabin's demonstration [13] of the remarkable effectiveness of probabilistic algorithms. Indeed a summary of the historical context of that work may be found in Morgan and McIver's text [8], to which we refer for an exposition of the distributional model and its relationship both to standard models and to the transformer model of nondeterminism and probabilism for sequential programs.

We have adopted the view—standard in Mathematics but less so in Computation where it has great relevance—that a conditional probability  $P(A \mid B)$  may be used to update knowledge about  $P(A)$  (for example by Bayes's formula) in the light of further, in our case sequentially provided, information. It thus forms the basis for a semantics of probabilistic programs. Naturally that view is not new. Incisive use of it has been made, notably, by Panangaden [12] and Ying [17].

Ying uses it as the basis for a semantics of guarded-command-like programs with angelic choice and (of course) demonic choice but without recursion, iteration or explicit probabilistic choice that has the strength to support a refinement calculus. His models extend the usual Boolean-valued models, of (a) expectation transformers and (b) sets of (sub) distributions, by considering instead probabilistic predicates. As a result his semantics makes finer distinctions between programs and he is able to introduce a refinement relation that is probabilistic rather than Boolean in nature. His semantics is based on a

definition of probabilistic implication, motivated by the definition of conditional probability. He extends many of the results of the standard theory, including the angelic-demonic factorisation theorem of Back and von Wright. Naturally his definition of sequential composition in the relational case reflects that of He *et al* [5].

Panangaden, on the other hand, makes a convincing case (based on the much earlier work of Givry) that conditional probability distributions are the counterpart—in general—of ‘probabilistic relations’. His treatment is aimed at the more general continuous case, but his insights apply here. It would be interesting to follow through his duality starting from our (nonstandard) state-based model to see what transformer model results. It would be interesting to apply, to the model proposed here, the ideas captured in those extensions to ‘probabilistic’ predicates or relations, both at the level of the types of our semantics and in the monadic setting. However here we consider only discrete state spaces. From Panangaden’s observation follows a whole approach to the analysis of information flow in security protocols; we mention just [2] which contains further references. Conditional distributions and conditional entropy are the driving forces behind that work.

Varacca and Winskel [16] give an elegant analysis of how the monads for nondeterminism and probability might be combined. They contrast the distributive combination of the two (used, for instance, by Mislove *et al.* [9]) with their combination after modifying the probabilistic monad to contain only affine identities (hence ensuring the result can be lifted to the power set). It would be extremely interesting to see how far that approach can be extended to refinement laws other than identities.

As indicated by our treatment of the Monty Hall problem and its variations, the topic of secrecy and information flow is implicit in our model. That is perhaps to be expected of any model containing both nondeterminism and probability, but only recently has the refinement setting been exploited to reason about security protocols. Morgan [11] uses a demonic-based framework to reason about Chaum’s ‘Dining Cryptographers’ and Rivest’s ‘Oblivious Transfer’. It would be interesting to extend his model to probability and, since that would require the use of conditional distributions, the approach of this paper may help in the relational setting. Chatzikokolakis and Palamidessi [1] use the process algebraic setting of a probabilistic variation of the pi-calculus to reason about partial secret exchange, again using ‘Oblivious Transfer’. Their refinement ordering is based on a probabilistic testing semantics. It would be of interest to see what observations and tests characterise refinement defined in our setting to be set containment.

There is a very much greater literature devoted to the difficult topic of reactive and parallel programs. See [15] for a survey, in the setting of probabilistic automata, to 2004. Inevitably some of those contributions are relevant to the sequential case. But in general that setting appears to be comprehensively more difficult as a result of reactivity. Typically there demonic nondeterminism is viewed as freedom to be exploited by a scheduler [14]. Whilst true as far as that is able to be exploited in the sequential case (for ‘scheduler’ read ‘implementer’), there it is not the overriding consideration, which is the interaction between probabilistic choice and sequential composition. Focusing on that, the sequential case might be viewed as a convenient starting point for a later study of reactive nondeterministic and probabilistic computation, in which the interaction between nondeterminism and probability is studied without the concerns of deadlock, divergence and so on. At present the two areas seem to be almost disjoint.

Contributions to the reactive case that will no doubt be influential due to the combination of nondeterminism and probabilism include the work of Mislove *et al.* [9] which constructs a model, of process algebra with nondeterminism and probabilism, as a solution to a domain equation using the Plotkin powerdomain. We mention the work [3] of Deng and Palamidessi as an important continuation of the influential school of de Bakker and his students. Based on ACP but incorporating ideas from the probabilistic automata of Segala [14], they find complete axiomatisations for finite state processes that contain both nondeterminism and probabilism, under both strong and weak behavioural (behavioural sensitive) equivalence (but restricted to guarded recursion in the weak case). Not surprisingly both Mislove *et al.* and Deng and Palamidessi use techniques that originate with Milner for axiomatising strong bisimulation for core CCS. Such techniques need to be extended to the asymmetric setting to account for the refinement here considered instead of equivalence.

## 5 Conclusion

This paper has introduced a relational probabilistic model containing both probabilistic choice and nondeterministic choice. The standard demonic nondeterministic choice is decomposed into two operators: one that performs convex closure and the other that performs nondeterministic closure. But the key novel idea has been the introduction of original states and the use of conditional probability distributions, conditioned on some state before the start of the (current) computation.

That clarifies what nondeterminism really does and facilitates further generalisations. One such is to generalise commands such as assignment and probabilistic choice to observe the original state. That allows a later computation to perform compensation back to the starting point of the closest nondeterministic closure. Another possibility is to strengthen the manner in which a nondeterministic choice can exploit further history by introducing more original states. Then a pure nondeterministic choice can act against the probabilistic choices at several points set by nested nondeterministic closures. Such generalisation become possible only after we explicitly reveal what nondeterministic choices really do, and have obvious application beyond the realm of probabilistic programming. One obvious probabilistic application is in the design of security protocols where the adversary can be regarded as nondeterministic if it is unknown whether it can observe certain information and take advantage of the observation.

## References

- [1] K. Chatzikokolakis and C. Palamidessi. A framework for analyzing probabilistic protocols and its application to the partial secrets exchange. *Theoretical Computer Science*, **389**(3):512–527, 2007.
- [2] K. Chatzikokolakis, C. Palamidessi and P. Panangaden. On the Bayes risk in information-hiding protocols. In *Proceedings of the 20th IEEE Computer Security Foundations. IEEE Computer Society*, 341–354, 2007.
- [3] Y. Deng and C. Palamidessi. Axiomatizations for probabilistic finite-state behaviours. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computation*

- Structures*. LNCS **3441**:110–124, Springer Verlag, 2005.
- [4] S. E. Dunne and W. J. Stoddart, editors. *UTP'06, First International Symposium on Unifying Theories of Programming*, LNCS, **4010**, Springer Verlag, 2006.
- [5] J. He and K. Seidel and A.K. McIver, Probabilistic models for the guarded command language, *Science of Computer Programming*, **28**(2):171–192, 1997.
- [6] J. He and J.W. Sanders. Unifying Probability. In *Unifying Theories of Programming 2006*. 173–199, 2006.
- [7] C.A.R. Hoare and J. He, *Unifying Theories of Programming*, Prentice Hall, 1998.
- [8] A.K. McIver and C.C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer Verlag Monographs in Computer Science, 2005.
- [9] M. Mislove, J. Ouaknine and J. Worrell. Axioms for probability and nondeterminism. *ENTCS* **96**:7–28, 2004.
- [10] C. Morgan and A. McIver and K. Seidel, Probabilistic Predicate Transformers, *ACM TOPLAS*, **18**(3):325–353, 1996.
- [11] C.C. Morgan. The shadow knows: Refinement of ignorance in sequential programs. In *Mathematics of Program Construction, LNCS 4014*:359–378, Springer Verlag, 2000.
- [12] P. Panangaden. Probabilistic relations. In: C. Baier, M. Huth, M.Z. Kwiatkowska, M. Ryan (eds.) *PROBMIV'98*, 59–74, 1998.
- [13] M.O. Rabin. Probabilistic algorithms. In J.F. Traub (ed.), *Algorithms and Complexity: New Directions and Recent Results*, 21–39. Academic Press, 1976.
- [14] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In *Proceedings of the 5th International Conference on Concurrency Theory. LNCS 836*:1–43, Springer Verlag, 1994.
- [15] A. Sokolova and E. de Vink. Probabilistic automata: system types, parallel composition and comparison. In *Validation of Stochastic Systems: A Guide to Current Research. LNCS 2925*:1–43, Springer Verlag, 2004.
- [16] D. Varacca and G. Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, **16**(1):87–113, 2006.
- [17] M. Ying. Reasoning about probabilistic sequential programs in a probabilistic logic. *Acta Informatica* **39**:315–389, 2003.