



ELSEVIER

Science of Computer Programming 28 (1997) 171–192

Science of
Computer
Programming

Probabilistic models for the guarded command language

He Jifeng*, K. Seidel, A. McIver

*Oxford University Computing Laboratory, Programming Research Group,
11 Keble. Road, Oxford OX1 3QD, UK*

Abstract

The two models presented in this paper provide two different semantics for an extension of Dijkstra's language of guarded commands. The extended language has an additional operator, namely probabilistic choice, which makes it possible to express randomized algorithms.

An earlier model by Claire Jones included probabilistic choice but not non-determinism, which meant that it could not be used for the development of algorithms from specifications. Our second model is built on top of Claire Jones' model, using a general method of extending a probabilistic cpo to one which also contains non-determinism. The first model was constructed from scratch, as it were, guided only by the desire for certain algebraic properties of the language constructs, which we found lacking in the second model.

We compare and contrast the properties of the two models both by giving examples and by constructing mappings between them and the non-probabilistic model. On the basis of this comparison we argue that, in general, the first model is preferable to the second. © 1997 Elsevier Science B.V.

Keywords: Probabilistic programming language; Semantics; Algebraic laws

1. Introduction

Dijkstra's language of guarded commands with its weakest precondition semantics [1] put reasoning about sequential imperative programs on a secure footing. The aim of this paper is to extend this language, its semantics, and formal reasoning to sequential randomized algorithms. Such algorithms differ from standard algorithms in that at some stage they make a random choice of action such as, for example, picking a random value to assign to a variable. The benefits of randomization in terms of efficiency and simplicity are well known, and are discussed for example in the recent and extensive survey by Gupta et al. [3].

To express random choice we add a probabilistic choice operator to the syntax of the language of guarded commands and give two alternative semantic models for this

* Corresponding author. E-mail: Jifeng.He@comlab.ox.ac.uk.

extended language. We then investigate their algebraic properties and the relationship between them.

A standard relational model of a non-deterministic language maps each state to a *set* of final states. This supports a notion of *refinement*, which permits an implementation to be more deterministic than its abstract specification. Our first model, which we call *relational* because it uses an analogous construction, defines the semantics of a program as a mapping from initial states to sets of probability distributions over final states. The multiplicity of distributions captures the non-deterministic aspect of a program.

The relational model was constructed from scratch, as it were, guided only by the desire for certain algebraic properties. We are interested in algebraic laws because they provide a way of reasoning about transformations of programs which is much simpler than reasoning using the semantics directly. This is particularly valuable for an incremental approach to program development, which may require transforming a program from a structure which clearly mirrors that of its specification to one which most efficiently exploits the architecture of the machine which will execute it. Ultimately such transformations may be carried out automatically by an optimizing compiler. An alternative motive for transforming a program might be the need to implement it using a restricted technology which supports only a subset of the language. Algebraic transformations may furthermore be used to verify aspects of compiler design, as was shown by the derivation of correct machine code for the construct of the guarded command language [5].

In contrast to the relational model, the second model arose out of a general method of extending, or *lifting*, a probabilistic cpo to one which also contains non-determinism. This method was developed by [8] and first used to construct a probabilistic concurrent process algebra by [11]. For the sequential programs we take the probabilistic cpo due to Jones and express non-determinism as sets of her programs. We will call this model the *lifted* model.

The difference between our two semantic models is explained and made mathematically precise by a mapping from the lifted onto the relational model, which we present in Section 5. The arrows in Fig. 1 indicate the existence of further mappings which connect the probabilistic models and the non-probabilistic model for the language of guarded commands, which we call the *standard* model for short. The mapping from the standard model to a probabilistic model is an *embedding* and the mapping from a probabilistic model to the standard model a *projection*. Provided that they respect the

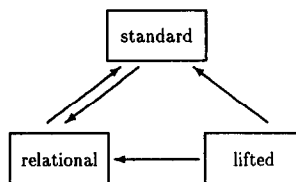


Fig. 1. Links between the models.

combinators of the standard language of guarded commands, both embedding and projection can be exploited to simplify reasoning about probabilistic programs. For example, a standard part of a probabilistic program could be proved correct using the (simpler) standard model and then be embedded in the probabilistic program in the knowledge that correctness was preserved. A projection could be used for proving non-probabilistic properties (typically safety properties) of probabilistic programs. We will show that all the mappings indicated in Fig. 1 have this property, and that the embedding-projection pair between the standard and the relational model form a so-called *Galois connection*. The latter is one of the reasons which will lead us to argue in the conclusion of the paper that the relational model is in general terms to be preferred to the lifted model.

2. Preliminaries

The language examined in this paper extends the guarded command language of Dijkstra [1] by including probabilistic choice $P_r \oplus Q$, which chooses between programs P and Q with probabilities r and $1-r$ respectively. In the following we will call all operators other than probabilistic choice *standard*. Their meaning will be explained in the next section. For now we define only the abstract syntax of the programming language.

Let P range over programs, b over Boolean expressions, and r over real numbers between 0 and 1. Assume that x stands for a list of distinct variables, and e for a list of expressions.

$$P ::= \text{ABORT} \mid \text{SKIP} \mid x := e \mid P \triangleleft b \triangleright P \mid \\ P_r \oplus P \mid P \oplus P \mid P; P \mid X \mid \mu X \bullet P(X)$$

A sequential program begins execution in an *initial* state and either terminates in a *final* state or fails to terminate. We model this by taking the set of all states to consist of the set S of *proper* states, which may be attained on termination, and the *improper* state \perp , which indicates non-termination. The set S is assumed to be countable. We write S_\perp for $S \cup \{\perp\}$.

For a proper state s we will use the notation $s; x := e$ for the state s with the value of its component variable x replaced by the value of the expression e .

So far we have referred to one standard model for the language of guarded commands where the meaning of a program is defined by a pair of predicates

$$(\text{pre}(s), \text{post}(s, s'))$$

where s and s' respectively denote the initial and final states of the program. If the program starts in an initial state satisfying the *precondition* pre , it will terminate in a final state satisfying the *postcondition* post . This is the approach used in the refinement calculus [9] and VDM [7]. Fig. 2 shows the semantics of the standard operators in this model. It uses composition of predicates defined by

$$Q(s, s'); R(s, s') \stackrel{\text{def}}{=} \exists t. Q(s, t) \wedge R(t, s').$$

$$ABORT \stackrel{def}{=} (false, true) \quad (1)$$

$$SKIP \stackrel{def}{=} (true, s' = s) \quad (2)$$

$$x := e \stackrel{def}{=} (true, s' = (s; (x := e))) \quad (3)$$

$$(pre1, post1) \oplus (pre2, post2) \stackrel{def}{=} (pre1 \wedge pre2, post1 \vee post2) \quad (4)$$

$$(pre1, post1) \triangleleft b \triangleright (pre2, post2) \stackrel{def}{=} (pre1 \triangleleft b \triangleright pre2, post1 \triangleleft b \triangleright post2) \quad (5)$$

$$(pre1, post1); (pre2, post2) \stackrel{def}{=} (pre1 \wedge \neg(post1; \neg pre2), post1; post2) \quad (6)$$

Fig. 2. The standard model.

A standard program *refines* another if it terminates more often and behaves less non-deterministically than the other. We write \sqsubseteq for “is refined by”; for programs $(pre1, post1)$ and $(pre2, post2)$ it is defined as

$$(pre1, post1) \sqsubseteq (pre2, post2) \stackrel{def}{=} \forall s. (pre1 \Rightarrow pre2) \wedge \forall s, s' \\ \cdot (pre1 \wedge post2 \Rightarrow post1).$$

We now turn to definitions which are necessary for the probabilistic models for the language of guarded commands. In all of these “.” denotes functional application.

Definition 2.1. A probability distribution f over S_{\perp} maps the subsets of S_{\perp} to the interval $[0, 1]$, such that

- (1) $f.\emptyset = 0$ and $f.S_{\perp} = 1$, and
 - (2) $f.\bigcup_{i \in I} X_i = \sum_{i \in I} f.X_i$ if $\{X_i \mid i \in I\}$ is a collection of disjoint subsets of S_{\perp} .
- Let D be the set of all probability distributions over S_{\perp} .

Definition 2.2. The point distribution η_s , for $s : S_{\perp}$, is the probability distribution defined as

$$\eta_s.X \stackrel{def}{=} \begin{cases} 1 & \text{if } s \in X \\ 0 & \text{otherwise} \end{cases}$$

We say that a probability distribution refines another if it assigns higher probabilities to all sets of proper states.

Definition 2.3. For probability distributions $f, g : D$

$$f \leq g \stackrel{def}{=} (\forall X \subseteq S \cdot f.X \leq g.X).$$

Clearly the relation \leq defines a partial order.

The following two definitions are crucial for our modelling of non-determinism and refinement, and were inspired by the construction of Smyth power domains [8] and the

definition of “indifferent” non-deterministic choice in [11]. Recall from the introduction that a non-deterministic program may lead to a set of distributions over final states if it is started in some proper initial state. We say that a program P refines a program Q if from every starting state, P results only in distributions which are refinements of distributions produced by Q . This definition is transitive only if the sets of distributions produced by any program are up-closed in the sense of the following definition.

Definition 2.4. A set T of probability distributions is *up-closed* if

$$T \supseteq \{g \mid \exists f \in T \cdot f \leq g\}.$$

We write $\uparrow T$ to denote the smallest up-closed set containing T .

Note that if $P.s$ is up-closed (for each P, s) the above concept of refinement between programs simplifies to set inclusion. We also want non-determinism to be refined by probabilistic choice, which, loosely speaking, is defined as the weighted average of its arguments. Thus we need non-determinism to include all the weighted averages of its arguments, i.e. to be convex-closed in the sense of Definition 2.5.

Definition 2.5. A set T of probability distributions is *convex-closed* if

$$T \supseteq T \oplus T$$

where

$$T_1 \oplus T_2 \stackrel{\text{def}}{=} \{r * f + (1 - r) * g \mid 0 \leq r \leq 1 \wedge f \in T_1 \wedge g \in T_2\}$$

We write $cc.T$ to denote the smallest convex-closed set containing T .

Both closure-operators are idempotent. We employ the following theorem from [8] to justify overlooking the order in which the closures are taken.

Theorem 2.6. For a set T of probability distributions

$$cc. \uparrow T = \uparrow cc.T.$$

3. The relational model

For our first model we take the view that the result of the execution of a program P starting at an initial state s can be described as a set of probability distributions over final states (including \perp). Thus programs are mappings of type

$$P : S_{\perp} \rightarrow \mathbb{P}D .$$

where $\mathbb{P}D$ denotes the collection of all subsets of D .

In the following we will consider any such mapping a valid specification, but the mappings which represent the semantics of programming language constructs will satisfy certain constraints. One such constraint is that we expect a program to behave chaotically if its initial state is improper, and the result of executing it could be any distribution at all. We therefore define the set *PROGS* of mappings with that property, namely

$$PROGS \stackrel{def}{=} \{P : S_{\perp} \rightarrow \mathbb{P}D \mid P.\perp = D\} .$$

Mappings in *PROGS* can be ordered by set inclusion: a program *P* in *PROGS* is worse (i.e. less determined) than a program *Q* if for every proper initial state *s*, *P* gives a bigger set of possible distributions over final states.

$$P \sqsubseteq Q \stackrel{def}{=} (\forall s \in S \cdot P.s \supseteq Q.s)$$

This makes (*PROGS*, \sqsubseteq) a complete lattice.

3.1. Semantics

We will associate each construct of our language with an element in *PROGS*, so for all programs *P*

$$P.\perp = D.$$

Definition 3.1 (*Semantics of the relational model*). Let *s* be a proper initial state. *ABORT* is the worst program, and its behaviour is totally unpredictable.

$$ABORT.s \stackrel{def}{=} D \tag{1}$$

Program *SKIP* terminates immediately, its final state being the same as its initial state.

$$SKIP.s \stackrel{def}{=} \{\eta_s\} \tag{2}$$

Let *x* be a list of distinct variables and *e* a list of expressions, and assume that evaluation of the expressions in *e* gives a list of values. The statement *x := e* assigns these values to the list of variables and then terminates.

$$(x := e).s \stackrel{def}{=} \{\eta_{s,x:=e}\} \tag{3}$$

Given *r* : [0, 1] we write *P* \oplus_r *Q* for the probabilistic choice between programs *P* and *Q*; they have probability *r* and 1 – *r* respectively of being selected.

$$(P \oplus_r Q).s \stackrel{def}{=} \{r * f + (1 - r) * g \mid f \in P.s \wedge g \in Q.s\} \tag{4}$$

The conditional construct $P \triangleleft b \triangleright Q$ is used to select the behaviour of a program depending on a predicate b of the initial values of its variables. It executes program P if b is true and program Q otherwise.

$$(P \triangleleft b \triangleright Q).s \stackrel{def}{=} \begin{array}{l} P.s \quad \text{if } b \text{ is true at state } s \\ Q.s \quad \text{otherwise} \end{array} \quad (5)$$

The program $P \oplus Q$ offers a non-deterministic choice between P and Q , which can be regarded as a probabilistic choice with unknown probability, because it is defined as the set of all probabilistic choices.

$$(P \oplus Q).s \stackrel{def}{=} P.s \oplus Q.s \quad (6)$$

We write $P;Q$ to denote the sequential composition of programs P and Q . Execution of P gives the set $P.s$ of distributions over intermediate states (which are invisible to the outside). Every distribution over the final states of $(P;Q).s$ is the result of picking for every intermediate state t a distribution in $Q.t$ and taking the weighted average of these distributions, the weights being the probabilities assigned to the intermediate states by some distribution in $P.s$.

$$(P;Q).s \stackrel{def}{=} \{\sum_{t:S_{\perp}} f.\{t\} * g_t \mid f \in P.s \wedge (\forall t : S_{\perp} \cdot g_t \in Q.t)\} \quad (7)$$

As usual, the recursion $\mu X \bullet P(X)$ is defined as the least fixed point of the monotonic function $P(X)$ in the complete lattice $(PROGS, \sqsubseteq)$.

$$\mu X \bullet P(X) \stackrel{def}{=} glb \{Y \in PROGS \mid Y \sqsupseteq P(Y)\} \quad (8)$$

where glb stands for the *greatest lower bound* operator in $PROGS$.

Theorem 3.2. *For any program P and any proper state s ,*

$$P.s = \uparrow cc.P.s$$

Proof. Assignment and *ABORT* are both up-closed and convex-closed. From the definition of the programming operators we conclude that they preserve both closure properties. It remains to show that the recursion $\mu X \bullet P(X)$ is also up-closed and convex-closed. Define for any $X \in PROGS$

$$L(X) \stackrel{def}{=} \lambda s. \uparrow cc.X.s$$

It is clear that $X \sqsupseteq L(X)$. From the definition it follows that X is both up-closed and convex-closed iff $L(X) = X$. In the following we are going to prove that

$$\begin{aligned}
 L(\mu X \bullet P(X)) &= \mu X \bullet P(X) \\
 &= L(\mu X \bullet P(X)) \\
 &= \{\text{unfold recursion}\} \\
 &= L(P(\mu X \bullet P(X))) \\
 &\sqsupseteq \{X \sqsupseteq L(X) \text{ and } L \text{ and } P \text{ are monotonic}\} \\
 &= L(P(L(\mu X \bullet P(X)))) \\
 &= \{\text{programming operators preserve both closure properties}\} \\
 &= P(L(\mu X \bullet P(X)))
 \end{aligned}$$

from which together with the definition of recursion we conclude

$$L(\mu X \bullet P(X)) \sqsupseteq \mu X \bullet P(X)$$

The inequality $\mu X \bullet P(X) \sqsupseteq L(\mu X \bullet P(X))$ is derived from the fact that $X \sqsupseteq L(X)$ for all $X \in PROGS$. \square

One of the key differences between the two models in this paper is the way non-determinism interacts with probabilistic choice. We will use the following example to illustrate this difference.

Example 3.3. Consider the two programs:

$$\begin{aligned}
 P &\stackrel{def}{=} x := 0 \oplus x := 1 \\
 Q &\stackrel{def}{=} y := 0 \text{ }_{0.5} \oplus y := 1 .
 \end{aligned}$$

Suppose that we are to execute P and Q sequentially in either order with the aim of establishing $(x = y)$. The execution can be regarded as a game against an adversary which uses the non-deterministic choices to work against us. If we execute Q after P then clearly whatever value the adversary chooses for x , y will be chosen independently of it and should agree with it with probability $1/2$.

This is confirmed by the semantics. Take s to be a proper state. Note that by the definition of non-deterministic choice we know that $P.s$ contains the point distributions $\eta_{s;x:=0}$ and $\eta_{s;x:=1}$. Also the set $Q.s$ contains only the single probability distribution q defined by

$$q.\{s; y := 0\} = q.\{s; y := 1\} = 0.5 .$$

Let $X = \{(x, y) : S \mid x = y\}$, that is the set of states in which x and y are equal. For $(P; Q).s$ we then have

$$\begin{aligned}
& \min \{f.X \mid f \in (P; Q).s\} \\
= & \{\text{def. of } (P; Q).s\} \\
& \min \{p.\{s; x := 0\} * q.\{(s; x := 0); y := 0\} + p.\{s; x := 1\} \\
& \quad * q.\{(s; x := 1); y := 1\} \mid p \in P.s\} \\
= & \min \{p.\{s; x := 0\} * 0.5 + p.\{s; x := 1\} * 0.5 \mid p \in P.s\} \\
= & 0.5.
\end{aligned}$$

If we execute Q before P , then the adversary can take advantage of the fact that the value of y is already determined, and always choose the opposing value of x . That is, we would expect that at worst $Q; P$ has zero probability of establishing $(x = y)$. This is borne out by the semantics:

$$\begin{aligned}
& \min \{f.X \mid f \in (Q; P).s\} \\
= & \min \{q.\{s; y := 0\} * p0.\{(s; y := 0); x := 0\} + q.\{s; y := 1\} \\
& \quad * p1.\{(s; y := 1); x := 1\} \mid p0 \in P.(s; y := 0) \wedge p1 \in P.(s; y := 1)\} \\
= & \{\text{taking } p0 = \eta_{s;x:=1} \text{ and } p1 = \eta_{s;x:=0}\} \\
& 0.
\end{aligned}$$

3.2. Algebraic laws

This section presents a set of algebraic laws which hold for the programming constructs defined in the previous section. Proofs that the laws are sound with respect to the semantics are straightforward and have been omitted. Here we will confine ourselves to those laws where there are differences between the relational model and the lifted model.

From the point of view of language design it is desirable to impose as few constraints as possible on the programming constructs, and make the laws as widely applicable as possible. Therefore we state the laws in such a way that they apply to any member of *PROGS* instead of just the programs expressible with the syntax of our language. In the following $cc.P$ ($P \in \text{PROGS}$) abbreviates $\lambda s.cc.P.s$ and similarly $\uparrow P$ abbreviates $\lambda s.\uparrow P.s$.

Conditional

Conditional choice is idempotent.

$$\mathbf{C-1} \quad P \triangleleft b \triangleright P = P.$$

Non-deterministic choice distributes over conditional choice.

$$\mathbf{C-2} \quad P \oplus (Q \triangleleft b \triangleright R) = (P \oplus Q) \triangleleft b \triangleright (P \oplus R).$$

Probabilistic Choice

Probabilistic choice is also idempotent, skew-symmetric and quasi-associative.

$$\mathbf{P-1} \quad cc.P \text{ } r \oplus cc.P = cc.P.$$

$$\mathbf{P-2} \quad P \text{ } r \oplus Q = Q \text{ }_{1-r} \oplus P.$$

$$\mathbf{P-3} \quad (P \text{ }_{r1} \oplus Q) \text{ }_{r2} \oplus R = P \text{ }_{s1} \oplus (Q \text{ }_{s2} \oplus R).$$

where $s1 = r1 * r2$ and $(1 - r2) = (1 - s1) * (1 - s2)$.

The choice can be eliminated if one argument is selected with probability 1.

$$\mathbf{P-4} \quad (P \oplus Q) = P.$$

It also distributes through conditional and non-deterministic choice.

$$\mathbf{P-5} \quad P \oplus (Q \triangleleft b \triangleright R) = (P \oplus Q) \triangleleft b \triangleright (P \oplus R).$$

$$\mathbf{P-6} \quad (P \oplus Q) \oplus cc.R = (P \oplus cc.R) \oplus (Q \oplus cc.R).$$

Sequence

Sequential composition distributes backwards through the choice operators.

$$\mathbf{S-1} \quad (P \oplus Q); cc.R = (P; cc.R) \oplus (Q; cc.R).$$

$$\mathbf{S-2} \quad (P \triangleleft b \triangleright Q); R = (P; R) \triangleleft b \triangleright (Q; R).$$

$$\mathbf{S-3} \quad (P \oplus Q); R = (P; R) \oplus (Q; R).$$

One might also expect the dual law to **P-6** to hold, but in this case we have only refinement, that is

$$(P \oplus Q) \oplus R \sqsupseteq (P \oplus R) \oplus (Q \oplus R)$$

The following example is a case where the refinement is strict.

Example 3.4. Compare the programs

$$P \stackrel{def}{=} (x := 1 \oplus_{0.5} x := 2) \oplus (x := 3)$$

$$Q \stackrel{def}{=} (x := 1 \oplus x := 3) \oplus_{0.5} (x := 2 \oplus x := 3).$$

Intuitively, in P the adversary chooses first and cannot exploit the outcome of the probabilistic choice, whereas in Q it is the other way round. Formally, the definition of probabilistic choice implies that the distributions over the final states of P must all give equal probability to $s; (x := 1)$ and $s; (x := 2)$, that is for any proper state s

$$f \in P.s \Rightarrow f.\{s; (x := 1)\} = f.\{s; (x := 2)\}$$

However, this is not true of Q because Q contains two separate non-deterministic choices, which can contribute different probabilities to the final distribution. Thus $Q.s$ contains for instance the distribution g where $g.\{s; x := 1\} = 0.25$, $g.\{s; x := 2\} = 0$ and $g.\{s; x := 3\} = 0.75$.

3.3. Link with standard model

This section investigates the connection between the relational model and the standard model. The idea is to find a pair of mappings which relate every program in the standard model to a program in the relational model and vice versa. In both directions we want the mappings to respect the combinators of the standard language of guarded commands, i.e. the semantics of a standard program text in the relational model should be mapped to the semantics of the same program text in the standard model and vice

versa. In the direction from relational to standard model we additionally want probabilistic choice to correspond to what is intuitively the closest construct in the standard model, namely non-deterministic choice. Clearly, going from the standard model to the relational model and back again should get us back to where we started, whereas going from the relational model to the standard model and back again we might end up with a program that is more non-deterministic than the original one. Formally, the pair of mappings should form a *Galois connection*.

The difference between the standard and the relational semantics is broadly speaking that the former tells us which final states are or are not possible, whereas the latter tells us the probability with which they may occur. To relate the latter to the former we take the view that a final state is possible if it has positive probability of occurring. More formally, the mapping \uparrow_1 projects a probabilistic program P onto the standard program $(pre(s), post(s, s'))$ as follows: $pre(s)$ is true, i.e. the standard program is guaranteed to terminate from initial state s , if and only if every distribution that P can reach from s has zero probability of non-termination; $post(s, s')$ is true, i.e. starting from s the standard program may end up in final state s' , if and only if s' is assigned a positive probability by at least one distribution that P can reach from s .

Definition 3.5. $\uparrow_1 P \stackrel{def}{=} (pre(s), post(s, s'))$ where

$$pre(s) = (\forall f \in P.s \cdot f.\{\perp\} = 0)$$

$$post(s, s') = (\exists f \in P.s \cdot f.\{s'\} > 0)$$

The following theorem states that as required, the mapping \uparrow_1 respects all the combinators of the standard programming language, and maps probabilistic choice to non-deterministic choice.

Theorem 3.6.

$$\uparrow_1 ABORT = (false, true). \quad (1)$$

$$\uparrow_1 SKIP = (true, s' = s). \quad (2)$$

$$\uparrow_1 x := e = (true, s' = (s; (x := e))) \quad (3)$$

$$\uparrow_1 (P \triangleleft b \triangleright Q) = (\uparrow_1 P) \triangleleft b \triangleright (\uparrow_1 Q). \quad (4)$$

$$\uparrow_1 (P \text{ } r \oplus \text{ } Q) \supseteq (\uparrow_1 P) \oplus (\uparrow_1 Q). \quad (5)$$

$$\uparrow_1 (P \oplus Q) = (\uparrow_1 P) \oplus (\uparrow_1 Q). \quad (6)$$

$$\uparrow_1 (P ; Q) = (\uparrow_1 P); (\uparrow_1 Q). \quad (7)$$

$$\uparrow_1 (\mu X \bullet P(X)) \supseteq \mu X \bullet (\uparrow_1 P)(X). \quad (8)$$

where $(\uparrow_1 P)(X)$ represents the term $P(X)$ after replacing all occurrences of the probabilistic choice operators $\text{ } r \oplus \text{ }$ in P by the non-deterministic choice operator \oplus .

Proof. In the following we use the notation $P.pre$ and $P.post$ to refer to the pre- and postcondition respectively of a standard process P . The first four cases are trivial. Case (5) is obvious if $r = 1$ or $r = 0$. If $0 < r < 1$ we have for the precondition

$$\begin{aligned}
& \uparrow_1 (P \oplus_r Q).pre(s) \\
= & \{\text{Def. 3.5}\} \\
& (\forall f \in (P \oplus_r Q).s \cdot f.\{\perp\} = 0) \\
= & \{\text{def. of } \oplus_r\} \\
& (\forall f \in P.s, g \in Q.s \cdot (r * f + (1 - r) * g).\{\perp\} = 0) \\
= & \{0 < r < 1\} \\
& (\forall f \in P.s, g \in Q.s \cdot f.\{\perp\} = 0 \wedge g.\{\perp\} = 0) \\
= & \{\text{Def. 3.5}\} \\
& (\uparrow_1 P).pre(s) \wedge (\uparrow_1 Q).pre(s)
\end{aligned}$$

as required. For the post-condition we have

$$\begin{aligned}
& \uparrow_1 (P \oplus_r Q).post(s, s') \\
= & \{\text{Def. 3.5}\} \\
& (\exists f \in (P \oplus_r Q).s \cdot f.\{s'\} > 0) \\
= & \{\text{Def 3.1 (4)}\} \\
& (\exists f \in P.s, g \in Q.s \cdot (r * f + (1 - r) * g).\{s'\} > 0) \\
= & \{0 < r < 1\} \\
& (\exists f \in P.s, g \in Q.s \cdot f.\{s'\} > 0 \vee g.\{s'\} > 0) \\
= & \{\text{Def. 3.5}\} \\
& (\uparrow_1 P).post(s, s') \vee (\uparrow_1 Q).post(s, s')
\end{aligned}$$

So if $0 < r < 1$ then case (5) can be strengthened to equality

$$\uparrow_1 (P \oplus_r Q) = (\uparrow_1 P) \oplus (\uparrow_1 Q), \quad \text{for } 0 < r < 1$$

Cases (6) and (7) can be proved similarly. It remains to prove the case for recursion.

$$\begin{aligned}
& \uparrow_1 (\mu X \bullet P(X)) \\
= & \{\text{unfold recursion}\} \\
& \uparrow_1 (P(\mu X \bullet P(X))) \\
\sqsupseteq & \{\uparrow_1 \text{ distributes through all programming operators except } \mu\} \\
& (\uparrow_1 P)(\uparrow_1 (\mu X \bullet P(X)))
\end{aligned}$$

The above and the least fixed point definition of recursion imply the conclusion (8). \square

The embedding of the standard model in the relational model, which we write as \Downarrow_b , is based on the following idea. Recall that if $pre(s)$ is true for some initial state s of a standard program, a final state s' is possible only if $post(s, s')$ is true. Probabilistically this can be expressed by saying that, starting from s , the program contains the distribution which assigns probability 1 to s' .

Definition 3.7.

$$\Downarrow_1(pre, post).\perp \stackrel{def}{=} D$$

$$\Downarrow_1(pre, post).s \stackrel{def}{=} cc.\{\eta_{s'} \mid post(s, s')\} \triangleleft pre(s) \triangleright D \quad \text{for } s : S.$$

Theorem 3.8. *The mappings \Uparrow_1 and \Downarrow_1 form a Galois connection.*

$$\Uparrow_1(\Downarrow_1(pre, post)) = (pre, post). \quad (1)$$

$$\Downarrow_1(\Uparrow_1 Q) \sqsubseteq Q. \quad (2)$$

Proof. Follows directly from the definitions. \square

The following theorem shows that \Downarrow_1 also respects the combinators of the standard language of guarded commands.

Theorem 3.9.

$$\Downarrow_1(false, true) = ABORT. \quad (1)$$

$$\Downarrow_1(true, s' = s) = SKIP. \quad (2)$$

$$\Downarrow_1(true, s' = (s; (x := e))) = (x := e) \quad (3)$$

$$\Downarrow_1(P \triangleleft b \triangleright Q) = (\Downarrow_1 P) \triangleleft b \triangleright (\Downarrow_1 Q). \quad (4)$$

$$\Downarrow_1(P \oplus Q) = (\Downarrow_1 P) \oplus (\Downarrow_1 Q). \quad (5)$$

$$\Downarrow_1(P; Q) = (\Downarrow_1 P); (\Downarrow_1 Q). \quad (6)$$

$$\Downarrow_1(\mu X \bullet P(X)) = \mu X \bullet (\Downarrow_1 P)(X). \quad (7)$$

Proof. Similar to Theorem 3.6. \square

4. The lifted model

In this section we construct the so-called *lifted* model for the guarded command language. It extends a model which is due to Jones [6] and which contains probabilistic choice but not non-deterministic choice.

Jones' semantics is based on mappings from proper states to *evaluations*. Broadly speaking, an evaluation is a function which is like a probability distribution except that it is defined only on certain sets and need not sum to 1. A precise definition can be found in [7], but for the special domain that we are dealing with, namely a flat, finite or countable state space, it turns out that the evaluations are defined on every set of states. The fact that an evaluation need not sum to 1 is used to model non-termination and to define an order on the evaluations. The same can be achieved in terms of probability distributions by adding the improper state \perp to the state space. This state is assigned the 'missing' probability, so that the distributions sum to 1, but is disregarded for the

order on distributions, which is based only on a comparison of the probabilities of proper states. This is the order given in Definition 2.3. Jones' semantics can therefore equivalently be cast in terms of mappings from states to probability distributions. We will take advantage of this, as it saves us from having to give yet more definitions and makes our exposition more homogeneous.

Thus a program in Jones' model is a mapping from states to probability distributions. Let P_D denote the set of such mappings.

$$P_D \stackrel{\text{def}}{=} S_{\perp} \rightarrow D$$

In the following we will call the programs in Jones' model *deterministic* because every initial state leads to exactly one probability distribution over final states. This information is insufficient to construct a non-deterministic operator, but allows the ordering \sqsubseteq over programs to be defined as the pointwise lifting of the ordering \leq over probability distributions: given $p, q : P_D$

$$p \sqsubseteq q \stackrel{\text{def}}{=} (\forall s : S \cdot p.s \leq q.s)$$

This notion of refinement turns the model into a *cpo* and thus into a candidate for the general techniques developed in [8] for constructing a power domain on top of domains with probability.

Following [11, 8], the model which we present in this section expresses non-determinism by taking convex-closed, up-closed sets of deterministic programs

$$P_L \stackrel{\text{def}}{=} \{P : \mathbb{P}(P_D) \mid P \text{ convex-closed and up-closed}\}$$

A *cpo* on this model is constructed using the Smyth ordering on convex-closed, up-closed sets, which in our case simplifies to inclusion-ordering: for $P, Q : P_L$ we define

$$P \sqsubseteq Q \stackrel{\text{def}}{=} P \supseteq Q$$

The binary operators in the lifted model are defined by their pointwise application of the Jones-operators, which we quote here.

Definition 4.1 (*Semantics of Jones' operators*). Let $p, q : P_D$ be deterministic programs, $s : S_{\perp}$ a state.

$$(p \oplus r \oplus q).s \stackrel{\text{def}}{=} r * p.s + (1 - r) * q.s \quad (1)$$

$$(p; q).s \stackrel{\text{def}}{=} \sum_{t: S_{\perp}} p.s.\{t\} * q.t \quad (2)$$

$$(p \triangleleft b \triangleright q).s \stackrel{\text{def}}{=} \begin{array}{l} p.s \quad \text{if } b \text{ is true at } s \\ q.s \quad \text{otherwise} \end{array} \quad (3)$$

4.1. Semantics

We can now define the semantics which extend Jones' model to include non-determinism.

Definition 4.2 (*Semantics of the lifted model*). The worst program, *ABORT*, is defined as the set of all deterministic programs.

$$ABORT \stackrel{def}{=} P_D \quad (1)$$

SKIP leaves any proper initial state unchanged; it is the singleton set containing the deterministic program which maps every initial state to the point distribution on that state. The up-closure ensures that starting from \perp any behaviour is possible.

$$SKIP \stackrel{def}{=} \uparrow \{\lambda s. \eta_s\} \quad (2)$$

Assignment $x := e$ also contains only a single deterministic program, namely the one which maps an initial state s to the point distribution $\eta_{s;x:=e}$, where we adopt the convention that $\perp; (x := e) = \perp$.

$$x := e \stackrel{def}{=} \uparrow \{\lambda s. \eta_{s;x:=e}\} \quad (3)$$

Sequential composition $P; Q$ is obtained by sequentially composing any two deterministic programs contained in P and Q respectively and taking the convex- and up-closure of the resulting set.

$$P; Q \stackrel{def}{=} cc. \uparrow \{p; q \mid p \in P \wedge q \in Q\} \quad (4)$$

Conditional choice and probabilistic choice are defined in the same manner as sequential composition, but they are automatically up-closed and convex-closed.

$$P \triangleleft b \triangleright Q \stackrel{def}{=} \{p \triangleleft b \triangleright q \mid p \in P \wedge q \in Q\} \quad (5)$$

$$P, r \oplus Q \stackrel{def}{=} \{p, r \oplus q \mid p \in P \wedge q \in Q\} \quad (6)$$

Finally we define the non-deterministic choice $P \oplus Q$ as the convex closure and up-closure of the union of sets P and Q .

$$P \oplus Q \stackrel{def}{=} cc. \uparrow (P \cup Q) \quad (7)$$

Recursion is defined simply as the least fixed point in the space of sets.

The following theorem, from [8], asserts the well-definedness of the above definitions.

Theorem 4.3. *Any program $P : P_L$ is a non-empty, up-closed and convex-closed set of deterministic programs, and satisfies $\{p. \perp \mid p \in P\} = D$.*

Intuitively speaking, non-determinism in the lifted model can be understood as a “compiler-time” decision, in the sense that it selects a deterministic program at the start of the execution (that is before any probabilistic choice has been made). By contrast, non-determinism in the relational model is a “runtime” decision: the choice of final distribution is made during the execution. This means that non-determinism in the relational model can take advantage of the outcome of all the probabilistic choices which have been made up to that point, but in the lifted model it cannot.

We illustrate this by returning to Example 3.3. Recall that $P \stackrel{\text{def}}{=} x := 0 \oplus x := 1$ and $Q \stackrel{\text{def}}{=} y := 0 \text{ }_{0.5} \oplus y := 1$. Let q denote the one deterministic program contained in Q . As before, if we execute P before Q , then P cannot take advantage of the value of x and $P; Q$ establishes $(x = y)$ with probability at least 0.5. Again let X be the set of states in which $x = y$.

$$\begin{aligned}
 & \min \{(p; q).s.X \mid p \in P\} \\
 = & \{\text{Def. 4.1 (2)}\} \\
 & \min \{\sum_{t \in S} p.s.\{t\} * q.t.X \mid p \in P\} \\
 = & \min \{p.s.\{s; x := 0\} * q.(s; x := 0).X + p.s.\{s; x := 1\} * q.(s; x := 1).X \mid p \in P\} \\
 = & \min \{p.s.\{s; x := 0\} * 0.5 + p.s.\{s; x := 1\} * 0.5 \mid p \in P\} \\
 = & 0.5
 \end{aligned}$$

However, if Q is executed before P , then the non-deterministic choice in P still cannot take advantage of the value of x because it has to pick one deterministic program in P to execute, and P only contains deterministic programs which are constant over all initial states.

$$P = \{\lambda s.(\eta_{s;x:=0} r \oplus \eta_{s;x:=1}) \mid 0 \leq r \leq 1\}$$

In the semantics of the lifted model, $Q; P$ therefore has probability at least 0.5 of establishing $(x = y)$:

$$\begin{aligned}
 & \min \{(q; p).s.X \mid p \in P\} \\
 = & \min \{\sum_{t \in S} q.s.\{t\} * p.t.X \mid p \in P\} \\
 = & \min \{q.s.\{s; x := 0\} * p.(s; x := 0).X + q.s.\{s; x := 1\} * p.(s; x := 1).X \mid p \in P\} \\
 = & \min \{0.5 * p.(s; x := 0).X + 0.5 * p.(s; x := 1).X \mid p \in P\} \\
 = & \min \{0.5 * r + 0.5 * (1 - r) \mid 0 \leq r \leq 1\} \\
 = & 0.5
 \end{aligned}$$

In other words, in the lifted model $Q; P = (Q; x := 0) \oplus (Q; x := 1)$: it is as if P 's choice was made before the execution of Q had started. In the relational model this is not true.

4.2. Algebraic properties

The lifted model shares many of the algebraic properties of the relational model; we shall point out only differences. We have already seen that sequential composition

distributes forward through non-deterministic choice in the lifted model, giving the new law

$$P; (Q \oplus R) = (P; R) \oplus (P; Q).$$

On the other hand it is no longer true that probabilistic choice distributes forwards through sequential composition: Law **S-3** has to be weakened, because on the left hand side R “knows” at compiler-time if it is to follow after P or Q and can therefore modify its behaviour accordingly, but on the right hand side this is not possible. Thus the behaviours of the right hand side are a subset of those of the left hand side.

$$(P \text{ } r \oplus \text{ } Q); R \sqsubseteq (P; R) \text{ } r \oplus \text{ } (Q; R)$$

A further difference is that conditional choice is no longer idempotent in the lifted model. To see this consider once more the program $P \stackrel{\text{def}}{=} x := 0 \oplus x := 1$ from Example 3.3. As mentioned above, P is state-insensitive in the sense that it contains only deterministic programs which choose probabilistically and independently of the initial state between 0 and 1. On the other hand, by definition of the conditional we have

$$P \triangleleft b \triangleright P = \{p1 \triangleleft b \triangleright p2 \mid p1, p2 \in P\}.$$

This allows different probabilistic choices depending on the value of the conditional. For instance it contains the deterministic program $\lambda s. \eta_{x:=0} \triangleleft b \triangleright \lambda s. \eta_{x:=1}$, which is sensitive to the initial state and is not contained in P . Thus law **C-1** is weakened to refinement:

$$(P \triangleleft b \triangleright P) \sqsubseteq P.$$

For similar reasons all the laws that distribute other operators over conditional choice, namely **C-2**, **P-5** and **S-2**, have turned into refinements:

$$\begin{aligned} (P \oplus Q) \triangleleft b \triangleright (P \oplus R) &\sqsubseteq P \oplus (Q \triangleleft b \triangleright R) \\ (P \text{ } r \oplus \text{ } Q) \triangleleft b \triangleright (P \text{ } r \oplus \text{ } R) &\sqsubseteq P \text{ } r \oplus \text{ } (Q \triangleleft b \triangleright R) \\ (P; R) \triangleleft b \triangleright (Q; R) &\sqsubseteq (P \triangleleft b \triangleright Q); R. \end{aligned}$$

4.3. Link with standard model

We can define a mapping \uparrow_2 from the lifted model to the standard model based on the same ideas as the mapping \uparrow_1 .

Definition 4.4.

$$\begin{aligned} (\uparrow_2 P).pre(s) &\stackrel{\text{def}}{=} (\forall p \in P \cdot p.s.\{\perp\} = 0) \\ (\uparrow_2 P).post(s, s') &\stackrel{\text{def}}{=} (\exists p \in P \cdot p.s.\{s'\} > 0) \end{aligned}$$

Theorem 4.5 states that \uparrow_2 respects the operators of the language of guarded commands. Its proof is similar to that of Theorem 3.6.

Theorem 4.5.

$$\begin{aligned}\uparrow_2 ABORT &= (false, true). \\ \uparrow_2 SKIP &= (true, s' = s). \\ \uparrow_2 (x := e) &= (true, s' = (s; (x := e))) \\ \uparrow_2 (P \triangleleft b \triangleright Q) &= (\uparrow_2 P) \triangleleft b \triangleright (\uparrow_2 Q). \\ \uparrow_2 (P \text{ ,}\oplus Q) &= (\uparrow_2 P) \oplus (\uparrow_2 Q) \quad \text{for } 0 < r < 1. \\ \uparrow_2 (P \oplus Q) &= (\uparrow_2 P) \oplus (\uparrow_2 Q). \\ \uparrow_2 (P; Q) &= (\uparrow_2 P); (\uparrow_2 Q). \\ \uparrow_2 (\mu X \bullet P(X)) &\sqsupseteq \mu X \bullet (\uparrow_2 P)(X).\end{aligned}$$

where $(\uparrow_2 P)(X)$ stands for the term $P(X)$ after replacing all occurrences of probabilistic choice operator $\text{ ,}\oplus$ by non-deterministic operator \oplus .

5. Linking the probabilistic models

We define a mapping Ψ from the lifted model to the relational model as follows.

Definition 5.1. For any program $P : P_L$ define

$$\Psi P.s \stackrel{def}{=} \{p.s \mid p \in P\}$$

From Theorem 4.3 it follows that ΨP lies in *PROGS*.

For each initial state, ΨP returns the set of all distributions which can be achieved by any of the deterministic programs in P . For different initial states, this need not mean execution of the same deterministic program in P . In this way Ψ turns the compiler-time non-determinism of the lifted model back into the runtime non-determinism of the relational model, as is asserted in the following theorem.

Theorem 5.2.

$$\begin{aligned}\Psi ABORT &= ABORT. & (1) \\ \Psi SKIP &= SKIP. & (2) \\ \Psi (x := e) &= (x := e). & (3) \\ \Psi (P \triangleleft b \triangleright Q) &= (\Psi P) \triangleleft b \triangleright (\Psi Q). & (4) \\ \Psi (P \text{ ,}\oplus Q) &= (\Psi P) \text{ ,}\oplus (\Psi Q). & (5)\end{aligned}$$

$$\Psi(P \oplus Q) = (\Psi P) \oplus (\Psi Q). \quad (6)$$

$$\Psi(P; Q) \sqsupseteq (\Psi P); (\Psi Q). \quad (7)$$

$$\Psi(\mu X \bullet P(X)) \sqsupseteq \mu X \bullet P(X). \quad (8)$$

Proof. For case (1) we have

$$\begin{aligned} & \Psi \text{ABORT}.s \\ = & \{\text{Def. 4.2 (1)}\} \\ & \{p.s \mid p \in P_D\} \\ = & \{\text{Def. of } P_D\} \\ & D \\ = & \{\text{Def. 3.1 (1)}\} \\ & \text{ABORT}.s \end{aligned}$$

The proofs for *SKIP* and assignment are similar. For case (4) we consider the branches of the conditional choice separately. Assume first that b is true at state s .

$$\begin{aligned} & \Psi(P \triangleleft b \triangleright Q).s \\ = & \{\text{Def. 4.2 (5)}\} \\ & \{(p \triangleleft b \triangleright q).s \mid p \in P \wedge q \in Q\} \\ = & \{\text{Def. 4.1 (3) and assumption}\} \\ & \{p.s \mid p \in P\} \\ = & \Psi P.s \end{aligned}$$

If b is false at state s we can show $\Psi(P \triangleleft b \triangleright Q).s = \Psi Q.s$ in a similar way. Cases (5) and (6) are straightforward. For (7) we have

$$\begin{aligned} & \Psi(P; Q).s \\ = & \{\text{Def. 4.2 (4)}\} \\ & cc. \uparrow \{(p; q).s \mid p \in P \wedge q \in Q\} \\ = & \{\text{Def. 4.1 (2)}\} \\ & cc. \uparrow \left\{ \sum_{t: S_{\perp}} p.s.\{t\} * q.t \mid p \in P, q \in Q \right\} \\ \subseteq & cc. \uparrow \left\{ \sum_{t: S_{\perp}} f.\{t\} * g_t \mid f \in \Psi P.s \wedge (\forall t: S_{\perp} \cdot g_t \in \Psi(Q)_t) \right\} \\ = & \{\text{Theorem 4.3, Def. 3.1 (7)}\} \\ & ((\Psi P); (\Psi Q)).s \end{aligned}$$

Case (8) is similar to Theorem 3.6 (8). \square

Note that clause (7) of Theorem 5.2 cannot be strengthened to equality. To show this recall the processes we considered in Example 3.3, namely

$$\begin{aligned} Q &= (y := 0)_{0.5} \oplus (y := 1) \\ P &= (x := 0) \oplus (x := 1). \end{aligned}$$

We show that $(\Psi Q); (\Psi P)$ can behave as a program which never sets $(x = y)$, but $\Psi(Q; P)$ cannot.

$$\begin{aligned}
& (\Psi Q); (\Psi P) \\
= & \{\text{Def. of } Q; \text{Theorem 5.2 (5)}\} \\
& (\Psi(y := 0)_{0.5} \oplus \Psi(y := 1)); \Psi P \\
= & \{\mathbf{S-3}\} \\
& (\Psi(y := 0); \Psi P)_{0.5} \oplus (\Psi(y := 1); \Psi P) \\
\sqsubseteq & \{\text{Theorem 5.2 (7)}\} \\
& \Psi(y := 0; P)_{0.5} \oplus \Psi(y := 1; P) \\
= & \{\text{Def. of } P; \text{distribution law of lifted model}\} \\
& \Psi(y := 0; x := 0 \oplus y := 0; x := 1)_{0.5} \oplus \Psi(y := 1; x := 0 \oplus y := 1; x := 1) \\
\sqsubseteq & \Psi(y := 0; x := 1)_{0.5} \oplus \Psi(y := 1; x := 0)
\end{aligned}$$

By contrast

$$\begin{aligned}
& \Psi(Q; P) \\
= & \{\text{Def. of } P, ; \text{ distributes forward through } \oplus \text{ in the lifted model}\} \\
& \Psi((Q; x := 0) \oplus (Q; x := 1)) \\
= & \{\text{Theorem 5.2(6)}\} \\
& \Psi(Q; x := 0) \oplus \Psi(Q; x := 1) \\
= & \{\text{Def. of } Q\} \\
& \Psi(y := 0; x := 1)_{0.5} \oplus \Psi(y := 1; x := 0)
\end{aligned}$$

However, if one is only interested in the possible final outcomes produced by the execution of a probabilistic program, i.e. in its projection onto the standard model, then the two probabilistic models give the same picture.

Theorem 5.3. *For any program P we have $\uparrow_1(\Psi P) = \uparrow_2(P)$.*

Proof. For the precondition,

$$\begin{aligned}
& \uparrow_1(\Psi P).pre(s) \\
= & \{\text{Def. 3.5}\} \\
& (\forall f \in (\Psi P).s \cdot f.\{\perp\} = 0) \\
= & \{\text{Def. 5.1}\} \\
& (\forall p \in P \cdot p.s.\{\perp\} = 0) \\
= & \{\text{Def. 4.4}\} \\
& (\uparrow_2 P).pre(s)
\end{aligned}$$

For the postcondition,

$$\begin{aligned}
& \uparrow_1(\Psi P).post(s, s') \\
= & \{\text{Def. 3.5}\} \\
& (\exists f \in (\Psi P).s \cdot f.\{s'\} > 0) \\
= & \{\text{Def. 5.1}\} \\
& (\exists p \in P \cdot p.s.\{s'\} > 0) \\
= & \{\text{Def. 4.4}\} \\
& (\uparrow_2 P).post(s, s') \quad \square
\end{aligned}$$

6. Conclusion

We have presented two semantic models for an extended language of guarded commands which contains both non-deterministic and probabilistic choice. The key difference between the two models lies in the way non-determinism interacts with probabilistic choice. If program execution is considered as a game against an adversary who decides the outcome of non-deterministic choices, then the difference of our two models concerns the point at which the adversary has to make his decision. In the relational model it is when non-deterministic choice is reached during execution, which means that the adversary can exploit the outcome of all probabilistic choices made up to that point. By contrast, in the lifted model the adversary must decide at the start of execution, and therefore cannot take advantage of any probabilistic choice going a particular way. We call this compiler-time non-determinism, as opposed to the runtime non-determinism of the relational model. Any probabilistic semantics assumes that non-deterministic and probabilistic choice can be distinguished by repeated tests: the former may behave totally unpredictably, whereas the latter should result in a frequency of outcomes which roughly mirror their probability. Runtime and compiler-time non-determinism are distinguishable only if the tester does not re-compile the program after each test.

Having presented two semantic models for the same language, the question arises which one should be the model of choice. In general it is up to the designer of a language to decide which algebraic properties it ought to have. This in turn depends on what is being modelled. For instance, some formalisms distinguish between non-determinism and underspecification: the latter presents the implementor with a choice from a set of alternatives, each of which is deterministic. Usually non-determinism allows more refinements than underspecification. The distinction between the two clearly is reflected in the distinction between runtime non-determinism (in our first model) and compiler-time non-determinism (in our second model). If implementation by deterministic programs is considered an important property, as it might be in security, then the second model would be more appropriate.

However, the lifted model lacks some very basic algebraic properties, such as idempotence of conditional choice. This lack actually motivated the search for another model. Another important consideration are the links between different models: the fact that the relational model is linked to the standard model by a Galois connection makes it possible to import algebraic proofs from the standard model to the probabilistic model, which is a significant practical advantage over the lifted model. We feel therefore that for general purposes the relational model would be more appropriate.

Most of the effort in modelling probabilistic systems to date seems to have been spent on concurrent systems, and very little on sequential systems and programming logics, with the exception of Fagin et al. [2] and Rao [12]. At the same time as these models presented here were being developed, Morgan et al. were working on a semantics for probabilistic predicate transformers which would extend Jones' probabilistic programming logic [10] with non-determinism. It turns out that the relational model

can be embedded in these much as the relations which model standard programs can be embedded in the standard predicate transformers.

Acknowledgements

The authors wish to acknowledge Tony Hoare, Carroll Morgan and Jeff Sanders for discussions and suggestions. We are also grateful for the constructive comments made by referees on an earlier version of this paper.

References

- [1] E.W. Dijkstra, *A Discipline of Programming* (Prentice-Hall, Englewood Cliffs, NJ, 1976).
- [2] R. Fagin, J.Y. Halpern and N. Meggido, A logic for reasoning about probabilities, *Inform. and Comput.* **87** (1990) 78–128.
- [3] R. Gupta, S.A. Smolka and S. Bhashar, On randomization in sequential and distributed algorithms, *ACM Comput. Surveys* **26** (1) (1994) 7–86.
- [4] C.A.R. Hoare, Unified theories of programming, Technical Report, Oxford University Computing Laboratory, 1994; available from <http://www.comlab.ox.ac.uk/oucl/users/tony.hoare/publications.html>.
- [5] C.A.R. Hoare, H. Jifeng and A. Sampaio, Normal form approach to compiler design, *Acta Inform.* **30** (1993) 701–739.
- [6] C. Jones, Probabilistic Non-determinism, Doctoral Thesis, Edinburgh University, 1990; also available as Technical Report ECS-LFCS-90-105.
- [7] C.B. Jones, *Systematic Software Development using VDM* (Prentice-Hall, Englewood Cliffs, NJ, 1986).
- [8] A.K. McIver and C.C. Morgan, Probabilistic power domains, in preparation.
- [9] C.C. Morgan, *Programming from Specifications* (Prentice-Hall, Englewood Cliffs, NJ, 1994) 2nd edition.
- [10] C.C. Morgan, A. McIver, K. Seidel and J.W. Sanders, Probabilistic predicate transformers, Technical Report PRG-TR-5-95, Programming Research Group, January 1995.
- [11] C.C. Morgan, A. McIver, K. Seidel and J.W. Sanders, Refinement oriented probability for CSP, Technical Report PRG-TR-12-94, Programming Research Group, August 1994.
- [12] J.R. Rao, Reasoning about probabilistic parallel programs, *ACM Trans. Programming Languages Systems* **16** (3) (1994).