

This is a repository copy of *Automated Reasoning for Probabilistic Sequential Programs with Theorem Proving*.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/178721/>

Version: Accepted Version

Proceedings Paper:

Ye, Kangfeng, Foster, Simon orcid.org/0000-0002-9889-9514 and Woodcock, Jim orcid.org/0000-0001-7955-2702 (2021) Automated Reasoning for Probabilistic Sequential Programs with Theorem Proving. In: Fahrenberg, Uli, Gehrke, Mai, Santocanale, Luigi and Winter, Michael, (eds.) Relational and Algebraic Methods in Computer Science - 19th International Conference, RAMiCS 2021, Proceedings. 19th International Conference on Relational and Algebraic Methods in Computer Science, RAMiCS 2021, 02-05 Nov 2021 Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) . Springer , FRA , pp. 465-482.

https://doi.org/10.1007/978-3-030-88701-8_28

Reuse

Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Automated Reasoning for Probabilistic Sequential Programs with Theorem Proving


Kangfeng Ye, Simon Foster, and Jim Woodcock

University of York, York, UK
firstname.lastname@york.ac.uk

Abstract. Semantics for nondeterministic probabilistic sequential programs has been well studied in the past decades. In a variety of semantic models, how nondeterministic choice interacts with probabilistic choice is the most significant difference. In He, Morgan, and McIver’s relational model, probabilistic choice refines nondeterministic choice. This model is general because of its predicative-style semantics in Hoare and He’s Unifying Theories of Programming, and suitable for automated reasoning because of its algebraic feature. Previously, we gave probabilistic semantics to the RoboChart notation based on this model, and also formalised the proof that the semantic embedding is a homomorphism, and revealed interesting details. In this paper, we present our mechanisation of the proof in Isabelle/UTP enabling automated reasoning for probabilistic sequential programs including a subset of the RoboChart language. With mechanisation, we even reveal more interesting questions, hidden in the original model. We demonstrate several examples, including an example to illustrate the interaction between nondeterministic choice and probabilistic choice, and a RoboChart model for randomisation based on binary probabilistic choice.

1 Introduction

In our previous work [1], we give a probabilistic semantics to RoboChart [2], a domain-specific language for robotics and distinctive in its support for automated verification, based on He, Morgan and McIver’s relational model [3]. The semantics of the model is the theory of designs in Hoare and He’s Unifying Theories of Programming (UTP) [4]. The model embeds standard designs in probabilistic designs through the weakest completion solution [3] which is defined on the weakest prespecification [5] and a forgetful function [1, 3]. In this paper, we present our mechanisation of the probabilistic semantics in Isabelle/UTP [6], an implementation of UTP in the Isabelle/HOL theorem prover [7].

The main contributions of this work include (1) the formalisation of the proof of the homomorphism for the embedding of sequential composition, which is not addressed in our previous work [1]; and (2) the theory of probabilistic designs in Isabelle/UTP for automated reasoning of probabilistic nondeterministic sequential programs. All definitions and theorems in this paper are mechanised and accompanying icons () link to corresponding repository artifacts.

The remainder of this paper is organised as follows. In Section 2, we present an implementation of a randomisation algorithm in RoboChart, based on a binary probabilistic choice. Section 3 briefly describes the syntax of *pGCL* [8] (that is used in our mechanisation) with a program for the algorithm (with recursion), and the relational semantics in UTP. In Section 4, we describe how to represent probability distributions, and define weakest prespecification and a forgetful function in Isabelle/UTP. We show the embedding is a homomorphism on the structure of standard programs (Section 5), probabilistic choice (Section 6), nondeterministic choice (Section 7), and sequential composition (Section 8). We demonstrate the automated reasoning with three examples in Section 9, review related work in Section 10, and conclude in Section 11.

2 RoboChart

We consider a randomisation algorithm that aims to choose an integer number from a set of integers with equal probability. Let the set be $[0..N)$, that is, integers from 0 to $(N - 1)$. We implement the algorithm in an operation `ChooseUniform` in RoboChart, as shown in Figure 1. The core of RoboChart is a subset of UML state machines that allows modelling of robotic controllers. We use this model as an example for automated reasoning about probabilistic programs.

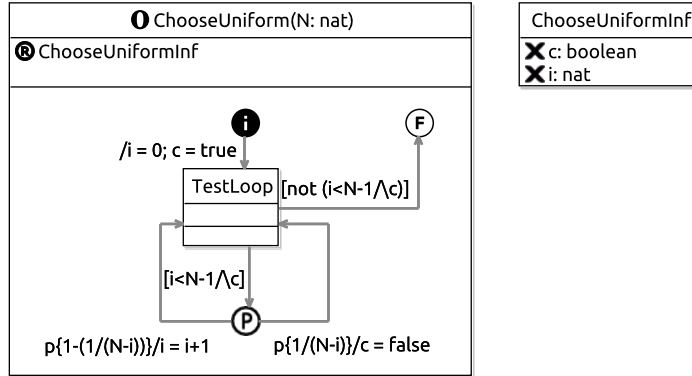


Fig. 1. A RoboChart model: uniform distribution algorithm.

The operation has one parameter N , denoting the size of the set, and has access (write and read) to two variables: c of type `boolean` and i of type `nat`, through the requested interface `ChooseUniformInf`. The operation is defined by a finite state machine having several nodes: one initial junction (●), a state `TestLoop`, a probabilistic junction (Ⓟ), and a final state (Ⓢ). Transitions connect nodes and are optionally labelled with a guard (a boolean expression e inside brackets, $[e]$), a probability value (an expression e inside braces after p , $p\{e\}$), and an action (a statement `act` after $/$, $/act$). This state machine, in general, implements the algorithm in three stages: (1) initialisation by the action of the

transition from \mathbf{i} to **TestLoop**; (2) iteration from **TestLoop** to \mathbf{P} , then back to **TestLoop**; and (3) termination by the transition from **TestLoop** to \mathbf{E} . The iteration is guarded by the condition if i has not reached $(N-1)$ and c is true. The termination is guarded by the negation of the condition. In each iteration, at \mathbf{P} , the state machine has probability $(1/(N-i))$ to update c to false (by the right transition) and so terminate next, and probability $(1 - 1/(N-i))$ to increase i (by the left transition) and so another iteration or terminate next depending on the condition ($i < (N-1)$). An accompanying tool for RoboChart ensures that the probabilities on outgoing transitions of a probabilistic junction add up to 1 through validation of well-formedness conditions of RoboChart models. If they do not add up to 1, then an error is displayed.

3 Probabilistic Programs

Syntax The abstract syntax of a nondeterministic probabilistic sequential programming language is given below.

$$P ::= \perp \mid \mathbf{I} \mid x := e \mid P \triangleleft b \triangleright Q \mid P \sqcap Q \mid P \oplus_r Q \mid P ; Q \mid \mu X \bullet P(X)$$

This probabilistic language introduces in the standard language a probabilistic choice operator $P \oplus_r Q$ which chooses between P and Q with probability r and $(1 - r)$ respectively. The syntax in the standard language includes abort \perp , skip \mathbf{I} , conditional $\triangleleft _ \triangleright$, and other common constructors.

The uniform distribution algorithm in Figure 1 is implemented as such a probabilistic program shown in Definition 3.1.

Definition 3.1 (The uniform distribution program).

ChooseUniform(N)

$$\triangleq i := 0 ; c := true ; \mu X \bullet \left(((c := false) \oplus_{1/(N-i)} (i := i + 1)) ; X \right) \triangleleft (i < (N - 1) \wedge c) \triangleright \mathbf{I}$$

Semantics The semantics of probabilistic programs is given in terms of probabilistic designs [1,3], being lifted from standard designs in UTP [4,9] via weakest completion semantics [3]. UTP employs Hehner's predicative style [10] to treat programs as predicates. It uses the alphabetised relational calculus to encode programs as relations between initial variable observations (x) and subsequent observations (x'). Relations are alphabetised predicates of which each is accompanied by its alphabet (a set of typed variable declarations). The alphabet of a predicate P is divided into the input alphabet ($in\alpha P = \{x, y, \dots\}$) and the output alphabet ($out\alpha P = \{x', y', \dots\}$).

UTP designs [4] are a subset of alphabetised predicates and denoted as $P \vdash R$: precondition-postcondition pairs. Designs have an additional variable ok in their alphabets to record the termination of programs. We use S to denote the state space of a program, containing only user variables and excluding ok . An alphabet induces a state space. Probabilistic designs are defined as $p(s) \vdash R(s, prob')$, where $s \in S$ and $prob' \in PROB$ (probabilistic state space). Here,

we use p instead of P , to denote that the precondition of a probabilistic design is a condition ($\text{out}\alpha p = \emptyset$). $PROB$ represents a set of (discrete) probability distributions over S : $PROB \triangleq S \rightarrow [0, 1]$. We are using discrete distributions because the only way to introduce them in the probabilistic programs is through the binary probabilistic choice operator. For each distribution $prob$ in $PROB$, its probabilities sum to 1: $\sum_{s \in S} prob(s) = 1$.

Both standard and probabilistic designs, which are used to give semantics to probabilistic programs, have their preconditions being conditions. In Isabelle/UTP, such designs are named normal designs [11] and denoted by $p \vdash_n R$.

Alphabetised predicates are presented in Isabelle/UTP through alphabetised expressions $[V, S]uexpr$, which are parametric over the value type V and the observation space S , and defined as total functions $S \rightarrow V$. Alphabetised predicates are boolean expressions: $[S]upred \triangleq [bool, S]uexpr$. Relations are predicates over a product space: $[S_1, S_2]urel \triangleq [S_1 \times S_2]upred$, where S_1 and S_2 are the initial and final observation space. Designs are relations with an additional ok variable: $[S_1, S_2]rel_des \triangleq [[S_1]des, [S_2]des]urel$, where des introduces the ok variable into alphabets. If S_1 is the same as S_2 , we use $[S]hrel_des$ for $[S, S]rel_des$.

4 Probabilistic Designs in Isabelle/UTP

We use the weakest completion solution [3] to embed standard designs D into probabilistic designs through an operator \mathcal{K} , where $\mathcal{K}(D) \triangleq D/\rho$, the weakest prespecification of ρ through D . The non-homogeneous design ρ (with alphabet $\{ok, prob, ok', s'\}$) is a forgetful function to retract states from probabilistic states and defined as $\rho \triangleq (\text{true} \vdash prob(s') > 0)$: the probability of arriving at state s' is $prob(s')$; this is replaced by the possibility ($prob(s') > 0$) of arriving at that state. So this function forgets the probability $prob$ in its initial observation space and retracts state s' in its final observation space. The embedding ($\mathcal{K}(D)$) is the weakest probabilistic design to make $\mathcal{K}(D); \rho$ a refinement of D .

We present the representation of probabilistic state spaces in Isabelle/UTP in Section 4.1. Then we describe our implementation of weakest prespecification and ρ in Isabelle/UTP in Section 4.2.

4.1 Representation of Probabilistic State Spaces

Isabelle/UTP provides a semantic framework for verification based on UTP, which is implemented in Isabelle/HOL, a generic proof system.

We use probability mass functions (PMFs) [12] in Isabelle/HOL to represent discrete probabilistic distributions. The type of PMFs ($[\alpha]pmf$, parametric over α) is a set of probability measure spaces. A measure space is a tuple $(\Omega, \mathcal{A}, \mu)$, where Ω is a set, \mathcal{A} is a σ -algebra on Ω , and μ is a measure function from \mathcal{A} to positive real numbers. A probability measure space is a measure space with its measure being 1 ($\mu(\Omega) = 1 < \infty$), and so finite.

We declare an observational variable $prob : [\alpha]pmf$ to represent distributions in probabilistic designs and use an **alphabet** command [6] to construct

a probabilistic state space, namely $[\alpha]prss$, a record type parametric over α : **alphabet** $[\alpha]prss = prob :: [\alpha]pmf$ (🍌). In the definition, $prob$ is a program variable whose type is $[\alpha]pmf$. Our probabilistic designs, therefore, are (non-homogeneous) designs: $[S_1, S_2]rel_pdes \triangleq [S_1, [S_2]prss]rel_des$, with initial observation space (state space S_1) to final observation space (probabilistic state space $[S_2]prss$ over state space S_2). They are non-homogeneous because the initial observation and final observation are over different spaces: state space versus probabilistic state space. For probabilistic programs defined in this paper, S_1 and S_2 are the same, and so probabilistic designs are actually homogeneous: $[S]hrel_pdes \triangleq [S, S]rel_pdes$ (🍌). Here, “homogeneous” means the final probabilistic observation space is over the same state space as the initial observation space. We define $[S]hrel_hpdes \triangleq [[S]prss, [S]prss]rel_des$ for homogeneous designs whose initial and final observation spaces are over probabilistic distributions.

4.2 Weakest Prespecification and Forgetful Function

Weakest prespecification is the generalisation of weakest precondition from a condition to a relation.

Definition 4.1. *The weakest prespecification of K through Y is defined as: $Y/K \triangleq \neg((\neg Y); K^-)$ (🍌) where $-$ is a relational converse operator.* 🍌

We note different notations are used for weakest prespecification in literature: $K \setminus Y$ in [5, 13] and Y/K in [3, 4]. Our mechanisation of weakest prespecification is based on [5] and so uses \setminus , but the mechanisation of probabilistic designs is based on [3]. We, therefore, use an abbreviation (🍌) to relate $/$ to \setminus . The weakest prespecification satisfies two theorems below.


Theorem 4.2. $Y \sqsubseteq (P; K) \Leftrightarrow (Y/K) \sqsubseteq P$ 🍌

This theorem shows that a program P is a refinement of the weakest prespecification of K through Y , if and only if a specification Y is implemented by sequential composition of P and K . The refinement relation $S \sqsubseteq P$ in UTP is defined as P implies S universally (for all alphabets of S and P): $S \sqsubseteq P \triangleq [P \Rightarrow S]$. For probabilistic designs, we rename predicates into $D \sqsubseteq (P; \rho) \Leftrightarrow (D/\rho) \sqsubseteq P$, which is interpreted as: a probabilistic design P implements the embedding of a standard design D into probabilistic designs through ρ if and only if the retraction of P through ρ to a standard design implements D . Another theorem is related to normal designs which our semantics relies on.

Theorem 4.3. $(p \vdash_n Q) / (\mathbf{true} \vdash_n R) = (p \vdash_n Q / R)$ 🍌

The weakest prespecification operator of two normal designs, when the precondition of its first design is **true**, can be moved into their postconditions.


The forgetful function ρ (**fp** in our mechanisation) is a non-homogeneous design, which is reflected in our definition below.

Definition 4.4. *The forgetful function $\mathbf{fp} : [[S]prss, S]rel_des$ is a normal design: $\mathbf{fp} \triangleq (\mathbf{true} \vdash_n \text{prob}(\mathbf{v}') > 0)$ where \mathbf{v} denotes all variables in the input alphabet of a program except the ok variable, and \mathbf{v}' are similar but in the output alphabet. We also say \mathbf{v} or \mathbf{v}' represent a state in state space.* 


From this definition, we know \mathbf{fp} is a relation between the initial probabilistic observation space and the final standard observation space.

5 Embedding Standard Programs


A design D is embedded into probabilistic designs through \mathcal{K} defined below.

Definition 5.1 (Embedding). $\mathcal{K}(D) \triangleq D / \mathbf{fp}$ 


Here, embedding is the weakest prespecification of \mathbf{fp} through D . So $\mathcal{K}(D)$ is the weakest probabilistic design related to D , and can be undone by retraction:

Theorem 5.2. *Let D be a normal design. $\mathcal{K}(D); \mathbf{fp} = D$* 

In other words, embedding a standard design into a probabilistic design, and then retracting returns the original design. Embedding is also monotonic:

Theorem 5.3. $P \sqsubseteq Q \Rightarrow \mathcal{K}(P) \sqsubseteq \mathcal{K}(Q)$ 

We show that embedded standard designs are probabilistic designs:

Theorem 5.4. *We fix the predicate $R : [S_1, S_2]urel$, then* 


$$\mathcal{K}(p \vdash_n R) = (p \vdash_n (\Sigma_a i \in S_2 \mid (R \mathbf{wp}(\mathbf{v} = i)) \bullet \text{pmf}(prob', i)) = 1)$$

Hence, embedding a standard normal design (LHS) is simplified to a normal design (RHS) with this theorem. Here, the form $(\Sigma_a x \in X \bullet \text{exp}(x))$ is a summation of the expression exp for all elements in set X . The symbol Σ_a denotes the summation over a possible infinite set. In the predicate $(R \mathbf{wp}(\mathbf{v} = i))$, \mathbf{wp} is the weakest precondition operator [14]. The predicate characterises the weakest precondition for R to be guaranteed to achieve $(\mathbf{v} = i)$. We recall that \mathbf{v} denotes all variables in state space, and so in other words, this predicate is simply a condition characterising when a given state i is a possible final state for R , which is equal to $\exists s \in S_1 \bullet R(s, i)$.

In the expression part of the summation, $\text{pmf}(prob', i)$, the function pmf returns the probability measure of the single state i in the distribution $prob'$. We use coercion in Isabelle/HOL to simplify its syntax further to $prob'(i)$. We also use $prob'(X)$ to denote $(\Sigma_a x \in X \bullet prob'(i))$, the probability measure of a set of states. We use the simpler syntax in the rest of the paper.

From the theorem, we know that the precondition p is unchanged after embedding. The postcondition is a condition such that the probabilities of all the final states of R sum to 1 and so $prob'$ is a distribution.

The embedding of abort, skip, assignment, and conditional is given in the theorem below.

Theorem 5.5. 

$$\mathcal{K}(\perp_D) = \perp_D \quad (\perp_D \triangleq \mathbf{false} \vdash \mathbf{false}) \quad (1)$$

$$\mathcal{K}(x :=_D e) = (\mathbf{true} \vdash_n \text{prob}'(\mathbf{v}[e/x]) = 1) \quad (x :=_D e \triangleq \mathbf{true} \vdash \mathbf{v}' = \mathbf{v}[e/x]) \quad (2)$$

$$\mathcal{K}(\mathbb{I}_D) = (\mathbf{true} \vdash_n \text{prob}'(\mathbf{v}) = 1) \quad (\mathbb{I}_D \triangleq \mathbf{true} \vdash \mathbf{v}' = \mathbf{v}) \quad (3)$$


$$\mathcal{K}(P \triangleleft b \triangleright Q) = \mathcal{K}(P) \triangleleft b \triangleright \mathcal{K}(Q) \quad (4)$$

Embedding the design abort (1, defined on the right) is still itself. Embedding an assignment $x :=_D e$ (2, defined on the right) is a probabilistic design with precondition **true** and postcondition establishing that the probability of the state with e substituted for x (the values of other variables are unchanged) is equal to 1, that is, embedding an assignment results in a point distribution: from each initial state \mathbf{v} , prob' in the final state has probability 1 for the state $\mathbf{v}[e/x]$. Based on the fact that prob' is a distribution, this also implies the probabilities for other states in prob' are 0. A design skip is a special form of assignment and so is its embedding (3). \mathcal{K} distributes through conditional, shown in (4).

6 Distributions Combination and Probabilistic Choice


In this section, we introduce an operator to combine probability distributions, and then use this operator to construct probabilistic choice.

6.1 Distribution Combinations

Definition 6.1 (Distributions combination). We fix $P : [S]pmf$, $Q : [S]pmf$, and $r : \mathbb{R}$, and define a distribution plus operator $+_r$ to merge two distributions P and Q based on the weight r : 

$$P +_r Q \triangleq \text{join_pmf}(\text{pmf_of_list}[(P, r), (Q, 1 - r)])$$

This combination is essentially a join from a distribution of type $[[S]pmf]pmf$ constructed from a list with two elements using pmf_of_list : the first element is a pair from P to weight r and the second one is a pair from Q to weight $1 - r$. This join flattens two distributions based on their measure functions. The distribution combination satisfies the theorem below.

Theorem 6.2. We fix $i : S$ and assume $r \in [0..1]$, then 

$$(P +_r Q) i = P(i) * r + Q(i) * (1 - r)$$

The probability of a particular state i in the combined distribution is a weighted sum of P and Q based on their weights r and $1 - r$.

This combination operator also satisfies several theorems below.

Theorem 6.3. 

$$\begin{array}{lll} P +_r Q = Q +_{(1-r)} P & \text{(quasi-commutative)} & P +_0 Q = Q \quad \text{(zero)} \\ P +_r P = P & \text{(idempotent)} & P +_1 Q = P \quad \text{(unit)} \end{array}$$

Because $+_r$ is **idempotent**, we know that the set of discrete distributions represented by $[S]pmf$ is convex-closed [15]: the combination of a distribution with itself with any weight is still in the set. The operator is also quasi-associative:

Theorem 6.4 (Quasi-associativity). *We fix $r_1, r_2, w_1, w_2 : \mathbb{R}$, and assume $w_1 \in [0..1]$, $w_2 \in [0..1]$, $(1 - w_1) * (1 - w_2) = (1 - r_2)$, and $w_1 = r_1 * r_2$. Then* 🍌

$$P +_{w_1} (Q +_{w_2} R) = (P +_{r_1} Q) +_{r_2} R$$

This will be used to prove associativity of probabilistic choice in Theorem 6.8.

6.2 Probabilistic Choice

As shown in [1], we use UTP's parallel-by-merge scheme [4, Chap.7], $P \parallel_M Q$, to define probabilistic choice. The two parallel programs P and Q share the same initial observation space \mathbf{v} , then establish their own final observation spaces individually as $0.\mathbf{v}'$ and $1.\mathbf{v}'$. A merge predicate M then describes how \mathbf{v} , $0.\mathbf{v}'$ and $1.\mathbf{v}'$ are merged. A conjunction of three separate programs is sequentially composed with M . The three programs include the two parallel programs P and Q , and one program to copy the initial observation space. The final observation spaces from P , Q , and the copy program, therefore, are referred to as $0.\mathbf{v}$, $1.\mathbf{v}$, and \mathbf{v} in M . We start with the definition of a distribution merge operator:

Definition 6.5. $\mathbf{PM}(r) \triangleq (prob' = 0.prob +_r 1.prob)$ 🍌

The merge predicate establishes that the final distribution is the combination of the distribution ($0.prob$) from the first program and the distribution ($1.prob$) from the second program with weight r . Probabilistic choice is defined below.


Definition 6.6 (Probabilistic choice). *We fix $P, Q : [S]hrel_pdes$, $r : \mathbb{R}$.* 🍌

$$P \oplus_r Q \triangleq \left(P \parallel_{\mathbf{PM}(r)}^D Q \right) \triangleleft r \in (0..1) \triangleright (Q \triangleleft r = 0 \triangleright (P \triangleleft r = 1 \triangleright \top_D))$$

The probabilistic choice is defined as a conditional:


- if r is not in the open interval $(0..1)$, the choice is defined as follows:
 - if r is equal to 0, the choice is Q ;
 - if r is equal to 1, the choice is P ;
 - if r is not 0 and 1, the choice is the design miracle \top_D : a miraculous or infeasible specification, defined as (**true** \vdash **false**).
- if r is in the open interval $(0..1)$, the choice is between P and Q by parallel-by-merge: a design parallel composition ($\parallel_{\mathbf{PM}(r)}^D \triangleq \parallel_{\mathbf{DM}(\mathbf{PM}(r))}$) using the merge predicate $\mathbf{PM}(r)$ to merge $prob$ and another design merge predicate \mathbf{DM} to merge ok .

We note that the definition of probabilistic choice is not simply a parallel composition between P and Q , but a conditional to characterise two special cases:


$r = 0$ and $r = 1$. This is because of the definition of parallel composition. Let $P = (p \vdash_n \mathcal{P})$ and $Q = (q \vdash_n \mathcal{Q})$, then 

$$P \parallel_{\mathbf{PM}(r)}^D Q = \left(\begin{array}{l} (p \vee (q \wedge \neg \text{Pre}(\mathcal{Q}))) \wedge \\ (q \vee (p \wedge \neg \text{Pre}(\mathcal{P}))) \end{array} \right) \vdash_n \left(\mathcal{P} \parallel_{\mathbf{PM}(r)} \mathcal{Q} \right)$$

Here $\text{Pre}(\mathcal{P})$ is the predicate characterising the domain of a UTP relation \mathcal{P} . We note that r is not in the precondition. The parallel composition, therefore, cannot be simplified to P or Q for the special cases as its precondition has to take the other operand Q or P into account. Zero and unit, however, are important algebraic properties for probabilistic choice, and so we define it as conditional. The probabilistic choice satisfies the theorems below.


Theorem 6.7 (Quasi-commutative, zero, unit). *We assume $r \in [0..1]$.* 

$$P \oplus_r Q = Q \oplus_{(1-r)} P \quad P \oplus_0 Q = Q \quad P \oplus_1 Q = P$$

Theorem 6.8 (Quasi-associativity). *We fix $r_1, r_2, w_1, w_2 : \mathbb{R}$, and assume $w_1, w_2 \in [0..1]$, $(1 - w_1) * (1 - w_2) = (1 - r_2)$, $w_1 = r_1 * r_2$, and that P , Q , and R are probabilistic designs. Then $P \oplus_{w_1} (Q \oplus_{w_2} R) = (P \oplus_{r_1} Q) \oplus_{r_2} R$* 

Probabilistic choice, therefore, is quasi-associative, with the weights adjusted as specified. This is basically the extension of quasi-associativity (Theorem 6.4) of the distribution combination $+_r$.

Probabilistic choice is also left-distributive and right-distributive over non-deterministic choice and conditional.

Theorem 6.9. *We assume $r \in [0..1]$, P , Q , and R are normal designs.* 

$$\begin{aligned} P \oplus_r (Q \sqcap R) &= (P \oplus_r Q) \sqcap (P \oplus_r R) && \text{(distl-nondeterminism)} \\ (Q \sqcap R) \oplus_r P &= (Q \oplus_r P) \sqcap (R \oplus_r P) && \text{(distr-nondeterminism)} \\ P \oplus_r (Q \triangleleft b \triangleright R) &= (P \oplus_r Q) \triangleleft b \triangleright (P \oplus_r R) && \text{(distl-conditional)} \\ (Q \triangleleft b \triangleright R) \oplus_r P &= (Q \oplus_r P) \triangleleft b \triangleright (R \oplus_r P) && \text{(distr-conditional)} \end{aligned}$$


Even though $+_r$ is **idempotent** (Theorem 6.3), \oplus_r is not idempotent in general. This is due to the parallel-by-merge scheme used in its definition. \parallel_M is not idempotent in general. Consider, for example, $P \oplus_r P$ for $r \in (0..1)$. The probabilistic choice is just $P \parallel_{\mathbf{PM}(r)}^D P$, according to its definition. Based on the definitions of the parallel-by-merge scheme [4, Chap.7] and the merge predicate, $P(s, 0.\text{prob}')$, $P(s, 1.\text{prob}')$, and $\text{prob}' = 0.\text{prob}' +_r 1.\text{prob}'$ are established. The parallel composition $P(s, 0.\text{prob}') \parallel_{\mathbf{PM}(r)}^D P(s, 1.\text{prob}')$ is equal to $P(s, \text{prob}')$ (and so idempotent) only if the probability distributions in the final observation space of P are convex-closed. If the final observation space of P is a single distribution (so P is deterministic), then $P \oplus_r P = P$ because a singleton set is convex-closed with respect to $+_r$. If, for example, the final observation space of P contains two distributions (so P is nondeterministic), then for any $0 < r < 1$, $P \oplus_r P \neq P$. We note that the set of distributions in the embedding of nondeterministic choice, as illustrated in Theorem 7.1, is convex-closed.

6.3 Merge Witness Distributions

In [1], we define two projections \mathcal{F} and \mathcal{G} for decomposition of a probabilistic program into the probabilistic choice of two subprograms. The functions \mathcal{F} and \mathcal{G} map a distribution $prob$ over a set of states S into a distribution $0.prob$ over a subset A of S and another distribution $1.prob$ over a subset B of S separately with $A \cup B = S$. They, therefore, are projections.


This decomposition is useful when implementing a probabilistic program with multiway probabilistic choice, such as in the Reactive Modules formalism [16], in this $pGCL$ (with only binary probabilistic choice). This has been illustrated in [1, Sect.7]. This decomposition is also useful to provide witnesses for the merge predicate, and the witnesses are indeed required later to prove Theorem 7.1, an important distribution theorem for nondeterministic choice, as demonstrated in its proof in [1, Sect.8].

We define the projection \mathcal{F} below. \mathcal{G} shares the same definition (but with different arguments in its applications).

Definition 6.10. We fix $A, B : [S]set$, and $p : [S]pmf$. 

$$\begin{aligned} \mathcal{F}(A, B, p) &\triangleq \text{measure_of} \\ &\quad (\text{space}(p), \text{sets}(p), \lambda C \bullet p(C \cap (A - B)) * \text{ratio}(A, B, p) + p(C \cap A \cap B)) \\ \text{ratio}(A, B, p) &\triangleq (p(B - A) + p(A - B)) / p(A - B) \end{aligned}$$


The result of \mathcal{F} is a measure space constructed by *measure_of* from the space of the probability measure space p (by *space*), the σ -algebra of p (by *sets*), and a measure function (a curried function). In this function, we use $A - B$ to denote set difference between A and B . Indeed \mathcal{F} defines a probability distribution:

Theorem 6.11. We fix $P : [S]pmf$ and $A, B : [S]set$, and assume $P(A \cup B) = 1$, $P(A - B) > 0$, and $P(B - A) > 0$, then $\text{prob_space}(\mathcal{F}(A, B, P))$. 

The constructed measure space by \mathcal{F} is a probability space (*prob_space*), that is, its measure sums to 1.

7 Nondeterministic Choice

In predicative programming, including UTP, nondeterministic choice is defined simply as $P \sqcap Q \triangleq P \vee Q$, and, therefore, $P \vee Q \sqsubseteq P$. This is reflected in the semantics of the embedding of nondeterministic choice. We show \mathcal{K} distributes through nondeterministic choice in the theorem below.


Theorem 7.1. We assume P and Q are normal designs. 

$$\mathcal{K}(P \sqcap Q) = (\prod r \in [0..1] \bullet (\mathcal{K}(P) \oplus_r \mathcal{K}(Q)))$$

This theorem demonstrates that an embedding of the nondeterministic choice of two standard designs P and Q is just the nondeterministic choice of the

probabilistic choices of the embeddings of P and Q in all possible ways (all possible weights from 0 to 1 inclusive). For the two special cases 0 and 1 for r , according to Theorem 6.7, they are just $\mathcal{K}(Q)$ and $\mathcal{K}(P)$ respectively.

In relational semantics for probabilistic programs here, nondeterministic choice is refined by probabilistic choice with any particular weight in $[0..1]$:


Theorem 7.2. *We assume $r \in [0..1]$, P and Q are normal designs.* 

$$\mathcal{K}(P \sqcap Q) \sqsubseteq \mathcal{K}(P) \oplus_r \mathcal{K}(Q)$$

So probabilistic choice refines nondeterministic choice, which is the most significant difference of this relational model from others.


8 Sequential Composition

In our previous work [1], we did not complete formalisation of the proof presented in [3] that \mathcal{K} is a homomorphism for sequential composition. In particular, a Kleisli lifting operator \uparrow [3, Def.3.11] is defined to lift a probabilistic design to a design taking $(ok, prob)$ to $(ok', prob')$. A probabilistic design, therefore, is able to be sequentially composed with this lifted design because sequential composition requires the output alphabet of the first operand to be equal to the input alphabet of the second operand (and so two probabilistic designs are not allowed to be sequentially composed). The lifting operator \uparrow is defined below.

Definition 8.1. $\uparrow P \triangleq kleisli_lift2(\lfloor pre_D(P) \rfloor_{<}, pre_D(P) \wedge post_D(P))$ 

The operator \uparrow has one parameter: a probabilistic design $P : [S]hrel_pdes$, and is defined using another auxiliary function $kleisli_lift2$. The first argument $(\lfloor pre_D(P) \rfloor_{<})$ to that function is the design precondition $(pre_D(P))$ of P with its output alphabet dropped (by $\lfloor _ \rfloor_{<}$), and so the argument is a condition. The second argument is the postcondition of P .

The $kleisli_lift2$, defined below, has two parameters: q of type $[S]upred$, and R of type $[S]hrel_pdes$, and characterises a homogeneous design of type $[S]hrel_hpdes$, whose initial and final observation spaces are both over probabilistic distributions.

Definition 8.2. 


$$\begin{aligned} & kleisli_lift2(q, R) \\ & \triangleq \left(\left(prob \left(\llbracket q \rrbracket_p \right) = 1 \right) \vdash \right. \\ & \quad \left. \exists Q \bullet \left(\left(\forall ss \bullet prob'(ss) = \Sigma_{at} \bullet prob(t) * (Q(t))(ss) \right) \wedge \right. \right. \\ & \quad \left. \left. \left(\forall s \bullet \left(\neg (prob(\mathbf{v}') > \mathbf{0} \wedge \mathbf{v}' = s); \right. \right. \right. \right. \\ & \quad \left. \left. \left. \left(\neg R; (\forall t \bullet prob(t) = (Q(s))(t)) \right) \right) \right) \right) \end{aligned}$$

Generally, this definition establishes that if the program starts in every possible state satisfying q , then it terminates with a distribution from that state which makes R hold for that state and that distribution.


The precondition of the definition of *kleisli_lift2* means that the initial observation space *prob* is a distribution over all states satisfying the predicate *q*, where $\llbracket q \rrbracket_p$ denotes extraction of the characteristic set of *q*.

The postcondition of the definition is an existential quantification over *Q*, a function of type $S \rightarrow [S]pmf$. The predicate part of $\exists Q$ is a conjunction of two predicates. The first predicate establishes that for any state *ss*, its probability in the final observation space *prob'* is equal to the summation of the products of the probability of each state *t* in the initial observation space *prob* and the probability of *ss* in the distribution $Q(t)$ corresponding to *t*. This predicate characterises *Q*. The second predicate establishes that for any state *s*, if its probability in the initial distribution *prob* is larger than 0, then *R* must be satisfied with its initial observation state *s* (the initial observation of *R* is \mathbf{v} , which is the same as \mathbf{v}' in the precedent predicate where \mathbf{v}' is just *s* because of $\mathbf{v}' = s$) and its final observation distribution *prob'* being a distribution $Q(s)$. This predicate characterises *R* based on *Q*.

We now define sequential composition of probabilistic designs.


Definition 8.3 (Sequential composition). $P ;_p Q \triangleq P ; \uparrow Q$. 

\mathcal{K} distributes through sequential composition in the theorem below.


Theorem 8.4. *We fix $P, Q : [S]hrel_des$ and assume *P* and *Q* are normal designs, and *S* is finite. Then $\mathcal{K}(P ; Q) = \mathcal{K}(P) ;_p \mathcal{K}(Q)$.* 

We note that the assumption, *S* is finite, is necessary to prove this theorem. This assumption is hidden in the original proof [3, Theorem 3.12] when giving the witness function $f(u, v)$, where a cardinality $\#$ is used.

The \uparrow satisfies the theorems below.

Theorem 8.5. *We assume *P* and *Q* are probabilistic designs.* 

$$\begin{aligned} \uparrow(\mathcal{K}(\mathbb{I}_D)) &= (\mathbf{true} \vdash_n \mathit{prob}' = \mathit{prob}) && \text{(skip)} \\ P ;_p \mathcal{K}(\mathbb{I}_D) &= P = \mathcal{K}(\mathbb{I}_D) ;_p P && \text{(left/right unit)} \\ P \sqsubseteq Q &\Rightarrow \uparrow P \sqsubseteq \uparrow Q && \text{(monotonic)} \\ (\uparrow P) &\text{ is a normal design} && \text{(normal design)} \end{aligned}$$

The lifted probabilistic skip is simply a *skip*, that is, its initial and final observation spaces are the same. The probabilistic skip is both a left unit and a right unit (*left/right unit*). The operator \uparrow is also *monotonic*. We note the definition of \uparrow is not a normal design (see Definition 8.2), and just a general design \vdash . We use it to ease type constraints in the definition. The operator \uparrow , however, is proved to be a *normal design* .


9 Examples

The first example illustrates the proof of a probabilistic program with sequential composition, conditional, and probabilistic choice from Hehner's work [17,

Sect. 6]. We define $P1$, $P2$, and $P3$ used in Theorem 9.1.

$$\begin{aligned} P1 &\triangleq (\mathcal{K}(x :=_D 0) \oplus_{1/3} \mathcal{K}(x :=_D 1)) \\ P2 &\triangleq (\mathcal{K}(x :=_D x + 2) \oplus_{1/2} \mathcal{K}(x :=_D x + 3)) \\ P3 &\triangleq (\mathcal{K}(x :=_D x + 4) \oplus_{1/4} \mathcal{K}(x :=_D x + 5)) \end{aligned}$$

In $P1$, x is assigned 0 or 1 with probability 1/3 or 2/3. $P2$ has the equal probability to increase x by 2 or 3. $P3$ increases x by 4 or 5 with probability 1/4 or 3/4. The semantics of the composition of $P1$, $P2$, and $P3$ is shown below.

Theorem 9.1. 

$$P1 ;_p (P2 \triangleleft x = 0 \triangleright P3) = \left(\mathbf{true} \vdash_n \left(\begin{array}{l} \text{prob}'[2/x] = 1/6 \wedge \text{prob}'[3/x] = 1/6 \wedge \\ \text{prob}'[5/x] = 1/6 \wedge \text{prob}'[6/x] = 1/2 \end{array} \right) \right)$$

The probabilistic program is equal to a normal design whose precondition is **true** and postcondition establishes that in the final observation space, the value of x is 2, 3, and 5 each with probability 1/6, and 6 with probability 1/2. The result is the same as that of [17].

The second example originates in [15] to illustrate how probabilistic choice interacts with nondeterministic choice in the relational semantics. It is also discussed in [17, Sect. 10] about nondeterminism. We define below P , a nondeterministic choice between x assigned to 0 and 1, and Q , a probabilistic choice between y assigned to 0 and 1.

$$P \triangleq (\mathcal{K}(x :=_D 0) \sqcap \mathcal{K}(x :=_D 1)) \quad Q \triangleq (\mathcal{K}(y :=_D 0) \oplus_{1/2} \mathcal{K}(y :=_D 1))$$

We now consider sequential composition of P and Q in either order with the aim of establishing $(x = y)$. If Q is after P , then

Theorem 9.2. 

$$P ;_p Q = \left(\mathbf{true} \vdash_n \left(\begin{array}{l} (\text{prob}'[0, 0/x, y] = 1/2 \wedge \text{prob}'[0, 1/x, y] = 1/2) \vee \\ (\text{prob}'[1, 0/x, y] = 1/2 \wedge \text{prob}'[1, 1/x, y] = 1/2) \end{array} \right) \right)$$

The theorem shows that the probability of establishing $(x = y)$ is 1/2: both alternatives of the disjunction in the postcondition have the same probability to establish $(x = y)$. Informally, the nondeterministic choice in P cannot take advantage of the value of x (because P is executed first), and, therefore, the probability of establishing $(x = y)$ in $(P; Q)$ is determined by Q , no matter which value of x is chosen. If P is after Q , then


Theorem 9.3. 

$$Q ;_p P = \left(\mathbf{true} \vdash_n \left(\begin{array}{l} (\text{prob}'[0, 0/x, y] = 1/2 \wedge \text{prob}'[0, 1/x, y] = 1/2) \vee \\ (\text{prob}'[1, 0/x, y] = 1/2 \wedge \text{prob}'[0, 1/x, y] = 1/2) \vee \\ (\text{prob}'[0, 0/x, y] = 1/2 \wedge \text{prob}'[1, 1/x, y] = 1/2) \vee \\ (\text{prob}'[1, 0/x, y] = 1/2 \wedge \text{prob}'[1, 1/x, y] = 1/2) \end{array} \right) \right)$$

The theorem demonstrates that the probability of establishing $(x = y)$ can be 0 (the second alternative of the disjunctions in the postcondition), 1/2 (the first and fourth alternative), and 1 (the third alternative). Informally, the nondeterministic choice in P can now take advantage of the value of x because y has been probabilistically determined in Q (so the probability of y being 0 or 1 in each alternative is 1/2). P , therefore, can choose (1) x opposite to the value of y , and so the probability of establishing $(x = y)$ is 0; (2) x the same as the value of y , and so the probability of establishing $(x = y)$ is 1; (3) x always 0, and so the probability of establishing $(x = y)$ is 1/2; and (4) x always 1, and so the probability of establishing $(x = y)$ is 1/2. The choice between the four cases is nondeterministic and represented as disjunctions in the theorem.


Hehner [17] describes four varieties of nondeterminism: angelic, demonic, oblivious, and prescient. For $Q ;_p P$, if P is an angelic choice, the result corresponds to the third alternative; if P is a demonic choice, the result corresponds to the second alternative; if P is an oblivious choice, the result corresponds to the first and the fourth alternative. $Q ;_p P$ in Theorem 9.3, therefore, is more abstract and can be refined into angelic, demonic, and oblivious choice. P defined above is not prescient, and so it does not know the future value of Q in $(P ;_p Q)$. The program, therefore, has probability 1/2 to achieve $x = y$, shown in Theorem 9.2.

The third example is the algorithm in Figure 1 and its probabilistic program in Definition 3.1. First, we find and define an invariant for the recursion.

Definition 9.4 (Invariant). 

$$\begin{aligned} & \text{ChooseUniform_inv}(N) \\ \triangleq & \left(\mathbf{true} \vdash_n \left(\left(\left(c \wedge i < (N - 1) \Rightarrow \left(\left(\forall j < (N - i - 1) \bullet \right. \right. \right. \right. \right. \right. \right. \right. \right. \right. \right. \left. \left. \left. \left. \begin{array}{l} \text{prob}'(\mathbf{v}[j + i, \text{false}/i, c]) = 1/(N - i) \\ \text{prob}'(\mathbf{v}[N - 1, \text{true}/i, c]) = 1/(N - i) \end{array} \right) \wedge \right) \right) \wedge \right) \wedge \left(\neg (c \wedge i < (N - 1)) \Rightarrow \text{prob}'(\mathbf{v}) = 1 \right) \right) \end{aligned}$$

The postcondition of the definition is a conjunction: (1) the first conjunct establishes that if c is true and i is less than $(N - 1)$, the value of i in the final state space \mathbf{v} is a uniform distribution in close interval $[i, N - 1]$ (each with probability $1/(N - i)$); (2) the second conjunct corresponds to the termination of the program, the probability of the final state being the same as the initial state \mathbf{v} is 1. *ChooseUniform_inv*(N) indeed is an invariant for the recursion.

Theorem 9.5 (Invariant). *We assume N is larger or equal to 1.* 

$$\text{ChooseUniform_inv}(N) \sqsubseteq (\mu X \bullet \text{ChooseUniformBody}(N, X))$$

Here *ChooseUniformBody* is an abbreviation for the body of the recursion in Definition 3.1. Finally, *ChooseUniform*(N) is proved to be a uniform distribution.

Theorem 9.6. *We assume N is larger or equal to 1.*



$$\left(\begin{array}{l} \text{true} \vdash_n \left((\forall j \bullet j < (N - 1) \Rightarrow (\text{prob}'(\mathbf{v}[j, \text{false}/i, c] = 1/N))) \wedge \right) \\ \text{prob}'(\mathbf{v}[(N - 1), \text{true}/i, c]) = 1/N \end{array} \right) \\ \sqsubseteq \text{ChooseUniform}(N)$$

The program $\text{ChooseUniform}(N)$ satisfies a contract, the left-hand side of \sqsubseteq , that (1) the probability of finally arriving a state, of which i is less than $(N - 1)$ and c is *false*, is $1/N$; and (2) the probability of finally arriving a state, of which i is $(N - 1)$ and c is *true*, is also $1/N$. We, therefore, conclude the program implements a uniform distribution given N .

10 Related Work

Hurd [18] developed a formal framework in High-Order Logic (HOL) [19], a predecessor of Isabelle/HOL [7], for modelling and verification of probabilistic algorithms using theorem proving. The work uses mathematical measure theory to represent probability space. Hurd et. al [20] also mechanised $pGCL$ in HOL based on the quantitative logic [21], enabling verification of partial correctness of probabilistic programs. Our mechanisation is also based on measure and probability theory in HOL of Isabelle and uses the same notation $pGCL$. We, however, mechanise the relational semantics of $pGCL$ in the theory of designs in UTP, enabling reasoning about total correctness.

Audebaud et. al [22] use the monadic interpretation of randomised programs for probabilistic distributions (instead of measure theory) and mechanise their work in the Coq theorem prover [23]. They consider only probabilistic choice (without nondeterminism) in a functional language with recursion, not in a non-deterministic probabilistic imperative program setting like us.

Cock [24] presents a shallow embedding of $pGCL$ with Isabelle/HOL for proof automation. The work is based on McIver and Morgan's interpretation of a $pGCL$ program as an expectation transformer from post-expectations to pre-expectations [25]. Its mechanisation uses real numbers (\mathbb{R}) in Isabelle/HOL as a type for probabilities, which improves automation. By contrast, we use measure theory and PMFs in Isabelle/HOL to encode probability distributions. Additionally, we base our formalisation on Isabelle/UTP (instead of the shallow embedding of Cock's work) which enables modelling at high-level abstraction and unification of semantics with other paradigms such as time and reactive systems (this is important in order to capture the semantics of RoboChart).

11 Conclusions

Previously, we gave the probabilistic semantics to RoboChart based on He, Morgan and McIver's relational model for $pGCL$, and formalised its proof. In this paper, we present a mechanisation of the proof in Isabelle/UTP to enable automated reasoning for probabilistic programs.

We use measure theory and probability mass functions in Isabelle/HOL to represent probability distributions. Our mechanisation shows that (1) PMFs are convex closed, (2) the probabilistic choice is not idempotent in general, and (3) embedding distributes through sequential composition for finite state space.

Based on the mechanisation, we use several examples to illustrate the automated reasoning, including the randomisation algorithm in RoboChart. We note this notation is general enough to capture other distributions, and not restricted to uniform distributions illustrated here.

As illustrated by the probabilistic nondeterministic programs in Theorems 9.2 and 9.3, computations of probabilistic programs are related to those of imperative programs. Probability information has become predicates over the *prob* variable and nondeterminism becomes disjunctions of these predicates, which, therefore, enables us to reason about probabilistic programs using general designs or relational facilities in UTP, such as contract-based reasoning [26]. This is the way in the relational model to tackle reasoning complexity introduced in probabilistic programs.

Our immediate future work is to lift probabilistic designs into UTP’s reactive theory to unify the semantics of reactive, time, and probability in RoboChart.

Acknowledgements. This work is funded by the EPSRC projects RoboCalc (Grant EP/M025756/1), RoboTest (Grant EP/R025479/1), and CyPhyAssure¹ (Grant EP/S001190/1). The icons used in RoboChart have been made by Sarfraz Shoukat, Freepik, Google, Icomoon and Madebyoliver from www.flaticon.com, and are licensed under CC 3.0 BY.

References

1. Woodcock, J., Cavalcanti, A., Foster, S., Mota, A., Ye, K.: Probabilistic semantics for robochart - A weakest completion approach. In Ribeiro, P., Sampaio, A., eds.: Unifying Theories of Programming - 7th International Symposium, UTP 2019, Dedicated to Tony Hoare on the Occasion of His 85th Birthday, Porto, Portugal, October 8, 2019, Proceedings. Volume 11885 of Lecture Notes in Computer Science., Springer (2019) 80–105
2. Miyazawa, A., Ribeiro, P., Li, W., Cavalcanti, A., Timmis, J., Woodcock, J.: Robochart: modelling and verification of the functional behaviour of robotic applications. *Softw. Syst. Model.* **18**(5) (2019) 3097–3149
3. He, J., Morgan, C., McIver, A.: Deriving probabilistic semantics via the ‘weakest completion’. In Davies, J., Schulte, W., Barnett, M., eds.: Formal Methods and Software Engineering, Berlin, Heidelberg, Springer Berlin Heidelberg (2004) 131–145
4. Hoare, C.A.R., He, J.: Unifying Theories of Programming. Prentice-Hall (1998)
5. Hoare, C.A.R., He, J.: The weakest prespecification. *Information Processing Letters* **24**(2) (1987) 127–132
6. Foster, S., Baxter, J., Cavalcanti, A., Woodcock, J., Zeyda, F.: Unifying semantic foundations for automated verification tools in Isabelle/UTP. *Sci. Comput. Program.* **197** (2020) 102510

¹ CyPhyAssure Project: <https://www.cs.york.ac.uk/circus/CyPhyAssure/>

7. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: a proof assistant for higher-order logic. Springer (2002)
8. McIver, A., Morgan, C.: Introduction to pGCL: Its logic and its model. In: Abstraction, Refinement and Proof for Probabilistic Systems. Springer (January 2005)
9. Woodcock, J., Cavalcanti, A.: A Tutorial Introduction to Designs in Unifying Theories of Programming. In Boiten, E.A., Derrick, J., Smith, G., eds.: Integrated Formal Methods, 4th International Conference, IFM 2004, Canterbury, UK, April 4-7, 2004, Proceedings. Volume 2999 of Lecture Notes in Computer Science., Springer (2004) 40–66
10. Hehner, E.C.R.: A Practical Theory of Programming. Texts and Monographs in Computer Science. Springer (1993)
11. Guttman, W., Möller, B.: Normal design algebra. J. Log. Algebraic Methods Program. **79**(2) (2010) 144–173
12. Hölzl, J., Lochbihler, A.: Probability Mass Function. Technical report isabelle.in.tum.de/library/HOL/HOL-Probability/Probability_Mass_Function.html.
13. C.A.R.Hoare, He, J.: The weakest prespecification. Technical Report PRG44, OUCL (June 1985)
14. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Commun. ACM **18**(8) (1975) 453–457
15. Jifeng, H., Seidel, K., McIver, A.: Probabilistic models for the guarded command language. Science of Computer Programming **28**(2-3) (1997) 171–192
16. Alur, R., Henzinger, T.A.: Reactive modules. Formal Methods Syst. Des. **15**(1) (1999) 7–48
17. Hehner, E.C.R.: Probabilistic predicative programming. In Kozen, D., Shankland, C., eds.: Mathematics of Program Construction, 7th International Conference, MPC 2004, Stirling, Scotland, UK, July 12-14, 2004, Proceedings. Volume 3125 of Lecture Notes in Computer Science., Springer (2004) 169–185
18. Hurd, J.: Formal verification of probabilistic algorithms. Technical report, University of Cambridge, Computer Laboratory (2003)
19. Gordon, M.J.C., Melham, T.F., eds.: Introduction to HOL: A theorem proving environment for higher order logic. Cambridge University Press (1993)
20. Hurd, J., McIver, A., Morgan, C.: Probabilistic guarded commands mechanized in HOL. Theor. Comput. Sci. **346**(1) (2005) 96–112
21. Morgan, C., McIver, A., Seidel, K.: Probabilistic predicate transformers. ACM Transactions on Programming Languages and Systems (TOPLAS) **18**(3) (1996) 325–353
22. Audebaud, P., Paulin-Mohring, C.: Proofs of randomized algorithms in Coq. Sci. Comput. Program. **74**(8) (2009) 568–589
23. The Coq development team: The Coq Proof Assistant. coq.inria.fr Accessed: 2021-05-20.
24. Cock, D.: Verifying Probabilistic Correctness in Isabelle with pGCL. In Cassez, F., Huuck, R., Klein, G., Schlich, B., eds.: Proceedings Seventh Conference on Systems Software Verification, SSV 2012, Sydney, Australia, 28-30 November 2012. Volume 102 of EPTCS. (2012) 167–178
25. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer (2005)
26. Ye, K., Foster, S., Woodcock, J. In: Compositional Assume-Guarantee Reasoning of Control Law Diagrams Using UTP. Springer International Publishing, Cham (2020) 215–254