A Probabilistic Theory of Designs Based on Distributions

Riccardo Bresciani and Andrew Butterfield[*]

Foundations and Methods Group,
Trinity College Dublin,
Dublin, Ireland
{bresciar,butrfeld}@scss.tcd.ie

Abstract. We present a theory of designs based on functions from the state space to real numbers, which we term *distributions*. This theory uses predicates, in the style of UTP, based on homogeneous relations between distributions, and is richer than the standard UTP theory of designs as it allows us to reason about probabilistic programs; the healthiness conditions H1–H4 of the standard theory are implicitly accounted for in the distributional theory we present. In addition we propose a Galois connection linkage between our distribution-based model of probabilistic designs, and the standard UTP model of (non-probabilistic) designs.

1 Introduction

The Unifying Theories of Programming (UTP) aims at a semantic framework where programs and specifications can be modelled as alphabetised relational predicates, capturing the semantic models normally used for their formal description [HJ98, DS06, But10, Qin10]: the advantage of this common framework is that of enabling formal reasoning on the integration of the different languages.

UTP relies on untyped predicate calculus: programs are expressed by means of logical predicates (programs are predicates! [Heh84, Hoa85]), and different theories can be given a UTP semantics, and here we focus on the theory of designs: this theory allows us to reason about the total correctness of programs from the perspective of what preconditions must be met in order to reach some given postconditions.

A challenging research question is how to add probability to the picture, and in particular how to integrate it in a framework where non-determinism is present: we use a framework based on distributions over the state space, where the predicates used involve a homogeneous relation between before- and after-distributions. This allows us to define a probabilistic theory of designs, which can handle programs where both probabilistic and non-deterministic choice co-exist.

This paper is structured as follows: we describe the background to UTP, with particular focus on the standard theory of designs in that framework (§2);

[*] The present work has emanated from research supported by Science Foundation Ireland grant 08/RFP/CMS1277 and, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero – the Irish Software Engineering Research Centre.

B. Wolff, M.-C. Gaudel, A. Feliachi (Eds.): UTP 2012, LNCS 7681, pp. 105–123, 2013.

introduce a probabilistic framework based on distributions over the state space (§3); present a probabilistic theory of designs in this new framework (§4) and discuss its application to a well-known example (§5); and conclude (§6).

2 Background

2.1 UTP

UTP uses second-order predicates to represent programs: they are used to express relations among a set of *observable variables* which constitute their alphabet.

Observable variables usually occur as both undecorated and decorated with a dash $'$: the former refer to states before the program starts (*before-states*), whereas the latter refer to the final states reached after the program has run (*after-states*).

For example, a program using two variables x and y might be characterised by having the set $\{x, x', y, y'\}$ as an alphabet, and the meaning of the assignment $x := y + 3$ would be described by the predicate

$$x' = y + 3 \wedge y' = y.$$

In effect UTP uses predicate calculus in a disciplined way to build up a relational calculus for reasoning about programs.

In addition to observations of the values of program variables, often we need to introduce observations of other aspects of program execution via so-called auxiliary variables. For example the theory of reactive programs uses three auxiliary variables — namely ok, $wait$, tr, ref — to keep track of information concerning the current program run, such as termination, reach of a stable state, refusals, . . .

A key notion in UTP is that of *healthiness conditions*: they are usually characterised as monotonic idempotent predicate transformers whose fixpoints characterise sensible (healthy) predicates. In other words they outlaw all arbitrary predicate calculus statements that describe predicates with no sense — an example is $\neg ok \Rightarrow ok'$, which describes a "program" that must terminate when not started.

This notion is closely related to that of *refinement*, defined as the universal closure[1] of reverse implication:

$$S \sqsubseteq P \mathrel{\hat=} [P \Rightarrow S]$$

Healthy predicates form a lattice under the ordering induced by the refinement relation.

The refinement calculus enables the derivation of an implementation P from a specification S: such derivation can be proven correct if P is a valid refinement of S.

[1] Square brackets denote universal closure, *i.e.* $[P]$ asserts that P is true for all values of its free variables.

Most UTP theories developed so far deal only with non-deterministic choice, nevertheless the introduction of a probabilistic choice operator is beneficial to many application requiring a quantitative approach, for example to evaluate reliability of programs.

Nevertheless some lines of research are moving along this direction. In [HS06] the authors present an approach to unification of probabilistic choice with standard constructs. They provide an example of how the laws of pGCL could be captured in UTP as predicates about program equivalence and refinement. However only an axiomatic semantics was presented, and the laws were justified via a Galois connection to an expectation-based semantic model.

The approach presented in [CS09] is that of decomposing non-deterministic choice into a combination of pure probabilistic choice and a unary operator that accounted for its non-deterministic behaviour . It is worth underlining a comment of theirs, on how still unsatisfactory theories are with respect to the issue of having probabilistic and demonic choice to coexist.

The UTP model described in [He10], which is used to give a UTP-style semantics to a probabilistic BPEL-like language, relates an initial state to a final probability distribution over states, rather than relating before-variables to corresponding after-variables of the same type.

We have previously presented an encoding of the semantics of the probabilistic guarded command language (pGCL) in the UTP framework [BB11, BB12]. This encoding captures pGCL programs as predicate-transformers, on predicates over probability before- and after-distributions.

In §3 we will present the underlying distributional framework, which we subsequently use in order to obtain a probabilistic theory of designs.

2.2 The Standard Theory of Designs

Now that we have given a general overview of the UTP framework, we are going to focus on the theory of designs and present its UTP semantics.

The theory of designs patches the relational theory, in the sense that predicates from the relational theory fail to satisfy the following equality:

$$\mathit{true} ; \mathcal{P} = \mathit{true}$$

In fact according to the relational theory true is a left identity of the sequential composition operator:

$$\begin{aligned} \mathit{true} ; \mathcal{P} &\equiv \exists \underline{v}_m \bullet \mathit{true}\{\underline{v}_m / \underline{v}'\} \;\wedge\; \mathcal{P}\{\underline{v}_m / \underline{v}\} \\ &\equiv \exists \underline{v}_m \bullet \mathit{true} \;\wedge\; \mathcal{P}\{\underline{v}_m / \underline{v}\} \\ &\equiv \exists \underline{v}_m \bullet \mathcal{P}\{\underline{v}_m / \underline{v}\} \end{aligned}$$

Which reduces to true if $\underline{v} \in \mathit{fv}(\mathcal{P})$, or to $\mathcal{P}$ otherwise.

This has disastrous consequences, as this enables us to show that a program can recover from a never-ending loop:

$$\mathit{true} * \mathit{skip} \equiv \mu X \bullet X \equiv \bot \equiv \mathit{true}$$

. . . which is surprising, to say the least.

The theory of designs uses an additional auxiliary variable ok (along with its dashed version ok') to record start (and termination) of a program.

A design (specification) is made of a precondition Pre that has to be met when the program starts, and if so the program establishes Post upon termination, which is guaranteed:

$$\mathit{ok} \land \mathit{Pre} \Rightarrow \mathit{ok}' \land \mathit{Post}$$

for which we use the following shorthand:

$$\mathit{Pre} \vdash \mathit{Post}$$

The semantics of the assignment $x := y + 3$ in this theory is the following:

$$\mathit{true} \vdash x' = y + 3 \land y' = y$$

(if started, it will terminate, and the final value of x will equal the initial value of y plus three, with y unchanged).

The behaviour of true with respect to sequential composition is the desirable one, as now we have:

$$\begin{aligned}
\mathit{true}; (\mathit{Pre} \vdash \mathit{Post}) &\equiv \mathit{true}; \mathit{ok} \land \mathit{Pre} \Rightarrow \mathit{ok}' \land \mathit{Post}\\
&\equiv \exists\, \mathit{ok}_m, \underline{v}_m \bullet \mathit{true}\{\mathit{ok}_m/\mathit{ok}'\}\{\underline{v}_m/\underline{v}'\} \ \land\ (\mathit{ok}_m \land \mathit{Pre}\{\underline{v}_m/\underline{v}\} \Rightarrow \mathit{ok}' \land \mathit{Post})\\
&\equiv \exists\, \mathit{ok}_m, \underline{v}_m \bullet \mathit{true} \ \land\ (\mathit{ok}_m \land \mathit{Pre}\{\underline{v}_m/\underline{v}\} \Rightarrow \mathit{ok}' \land \mathit{Post})\\
&\equiv \exists\, \mathit{ok}_m, \underline{v}_m \bullet \mathit{ok}_m \land \mathit{Pre}\{\underline{v}_m/\underline{v}\} \Rightarrow \mathit{ok}' \land \mathit{Post}\\
&\equiv \mathit{true}
\end{aligned}$$

and therefore true is a left zero for sequential composition.

Designs form a lattice, whose bottom and top elements are respectively:

$$\mathit{abort} \mathrel{\hat{=}} \mathit{false} \vdash \mathit{false} \equiv \mathit{false} \vdash \mathit{true}$$

and

$$\mathit{miracle} \mathrel{\hat{=}} \mathit{true} \vdash \mathit{false} \equiv \neg \mathit{ok}$$

It should be noted that $\mathit{miracle}$ is a (infeasible) program that cannot be started.

Valid designs are predicates R which comply with four healthiness conditions [HJ98]. The first one (*unpredictability*, H1) excludes from observation all programs that have not started, and therefore restricts valid relations to those such that:

$$R = (\mathit{ok} \Rightarrow R)$$

All H1-healthy predicates satisfy the left zero and left unit laws:

$$\mathit{true}; R = \mathit{true} \qquad \text{and} \qquad \mathit{skip}; R = R$$

The second one (*possible termination*, H2) states that a valid relation cannot require nontermination:

$$R\{\mathit{false}/\mathit{ok}'\} \Rightarrow R\{\mathit{true}/\mathit{ok}'\}$$

The third one (*dischargeable assumptions*, H3) states that preconditions cannot use dashed variables. All H3-healthy predicates satisfy the right unit law:

$$R ; \mathit{skip} = R$$

The fourth one (*feasibility* or *excluded miracle*, H4) requires the existence of final values for the dashed variables that satisfy the relation:

$$\exists \mathit{ok}', \underline{v}' \bullet R = \mathit{true}$$

H4 excludes $\mathit{miracle}$ from the valid designs, and this implies that all H4-healthy predicates satisfy the right zero law:

$$R ; \mathit{true} = \mathit{true}$$

This condition cannot be expressed as an idempotent healthiness transformer, and does not preserve the predicate lattice structure. It serves solely to identify and/or eliminate predicates that characterise infeasible behaviour.

Through our distributional framework (§4) we obtain a richer theory where corresponding healthiness conditions hold (§4.1), even without the introduction of the auxiliary variables $\mathit{ok}, \mathit{ok}'$. Moreover the use of distributions enables us to evaluate the probability both of termination and of meeting a set of arbitrary postconditions as a function of the initial distribution (which determines the probability of meeting any required precondition).

3 The Distributional Framework

In [BB11] we have presented a UTP framework to deal with demonic probabilistic programs.

This framework relies on the concept of *distributions* over the state space: a generic distribution is a real-valued function $\chi : \mathcal{S} \to \mathbb{R}$ that assigns a *weight* x_i (a real number) to each state σ_i in the state space $\mathcal{S}$. The mathematics we employ is valid provided the probabilities constitute what is known as a measure space [Hal50]. If the state is not finite, then the limitation we face is that property predicates (pre- and post-expectations, for example) can only talk about probabilities associated with sets of observations, rather than single ones — in effect χ has to be interpreted as a probability density function.

A state σ bundles all the information regarding program variables into a single observation, in a style shared with many presentations of *Circus*-like languages: program variable values are modelled with a single state observation $\sigma : \mathcal{V} \to \mathcal{W}$, which is treated as a finite map from variables ($\mathcal{V}$) to values ($\mathcal{W}$). This choice simplifies the treatment of alphabets to a considerable degree.

The weight of a distribution is defined as:

$$\|\chi\| \mathrel{\hat=} \sum_{\sigma \in \operatorname{dom} \chi} \chi(\sigma)$$

This operation can be lifted to a set $\mathcal{X}$ of distributions:

$$\|\mathcal{X}\| \mathrel{\hat{=}} \{\|\chi\| \mid \chi \in \mathcal{X}\}$$

Among all generic distributions, the following two sub-classes play important roles in our framework:

- a *weighting distribution* π has the property that for every state σ we have $0 \leq \pi(\sigma) \leq 1$ — we define two particular weighting distributions, ϵ and ι, as the ones mapping every state to 0 and 1 respectively. There is no limit for the distribution weight;
- a *probability distribution* δ is a weighting distribution with the additional property that $\|\delta\| \leq 1$.

We will use the term *sub-distribution* to refer to a probability distribution where $\|\delta\| < 1$ and the term *full distribution* to refer to a probability distribution where $\|\delta\| = 1$.

Generally speaking, it is possible to operate on distributions by lifting pointwise operators such as addition, multiplication and multiplication by a scalar. Analogously we can lift pointwise all traditional relations and functions on real numbers[2].

In the case of pointwise multiplication, it is interesting to see it as a way of "re-weighting" a distribution. We have a particular interest in the case when one of the operands is a weighting distribution π, as we will use this operation to give semantics to choice constructs. We opt for a postfix notation to write this operation, as this is an effective way of marking when pointwise multiplication happens in the operational flow: for example if we multiply the probability distribution δ by the weighting distribution π, we write this as $\delta\langle\pi\rangle$. We use notation ϵ and ι to denote the everywhere zero and unit distributions, respectively:

$$\epsilon(\sigma) = 0 \wedge \iota(\sigma) = 1, \qquad \text{for all } \sigma$$

Given a condition (predicate on state) c, we can define the weighting distribution that maps every state where c evaluates to *true* to 1, and every other state to 0: as the value of each state can be seen as the boolean value of c in that state multiplied by 1, we overload the above notation and note this distribution as $\iota\langle c\rangle$. In general whenever we have the multiplication of a distribution by $\iota\langle c\rangle$, we can use the postfix operator $\langle c\rangle$ for short, instead of using $\langle\iota\langle c\rangle\rangle$. It is worth pointing out that if we multiply a probability distribution δ by $\iota\langle c\rangle$, we obtain a distribution whose weight $\|\delta\langle c\rangle\|$ is exactly the probability of being in a state satisfying c.

3.1 Assignment

A challenge we have faced has been describing how assignment, which is very much oriented towards individual variables, is given a semantics in terms of a

[2] Distributions form a vector space, which we have explored elsewhere. We omit discussion of this aspect of our theory for clarity and brevity.

distribution that involves complete entanglement of those variables. In effect an assignment statement $x := e$ involves a partial entanglement of variable x with the variables mentioned in e. In general as we build up larger programs using single assignment as the basic component we observe an increasing degree of entanglement, which can often be captured as an appropriate simultaneous assignment, so we shall work at this level here.

Given a simultaneous assignment $\underline{v} := \underline{e}$, where underlining indicates that we have lists of variables and expressions of the same length, we denote its effect on an initial probability distribution δ by $\delta\{\!|\underline{e}/\underline{v}|\!\}$. The postfix operator $\{\!|\underline{e}/\underline{v}|\!\}$ reflects the modifications introduced by the assignment — the intuition behind this, roughly speaking, is that all states σ where the expression $\underline{e}$ evaluates to the same value $\underline{w} = \mathrm{eval}_\sigma(\underline{e})$ are replaced by a single state $\sigma' = (\underline{v} \mapsto \underline{w})$ that maps to a probability that is the sum of the probabilities of the states it replaces.

$$(\delta\{\!|\underline{e}/\underline{v}|\!\})(\sigma') \mathrel{\hat{=}} (\textstyle\sum \delta(\sigma) \;\mid\; \sigma' = \sigma \dagger \{\underline{v} \mapsto \mathrm{eval}_\sigma(\underline{e}))$$

Here we treat the state as a map, where † denotes map override; this operator essentially implements the concept of "push-forward" used in measure theory, and is therefore a linear operator. An example is given in Figure 1.

Assignment preserves the overall weight of a probability distribution if $\underline{e}$ can be evaluated in every state, and if not the assignment returns a sub-distribution, where the "missing" weight accounts for the failing on some states (this failure prevents a program from proceeding and causes non-termination).

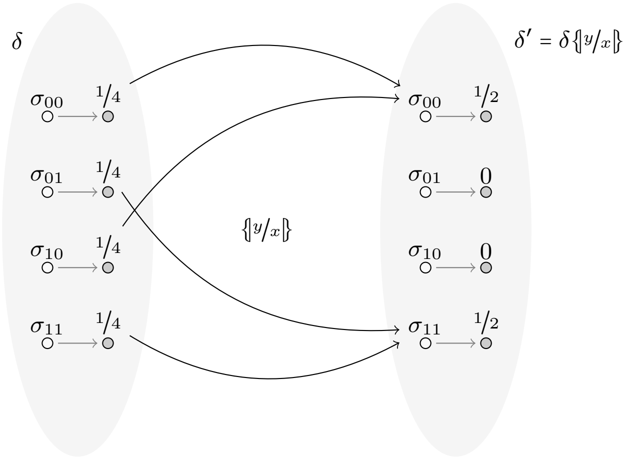

Fig. 1. The assignment $x := y$ from an initial uniform distribution on the state space $\mathcal{S} = \{0,1\} \times \{0,1\}$

$$\begin{aligned}
\mathit{abort} &\triangleq \|\delta'\| \leq \|\delta\| \\
\mathit{miracle} &\triangleq (\delta = \epsilon) \land (\delta' = \epsilon) \\
\mathit{skip} &\triangleq \delta' = \delta \\
\underline{v} := \underline{e} &\triangleq \delta' = \delta\{\!|\underline{e}/\underline{v}|\!\} \\
A ; B &\triangleq \exists \delta_m \bullet A(\delta, \delta_m) \land B(\delta_m, \delta') \\
\mathit{choice}\big(A, B, \mathcal{X}\big) &\triangleq \exists \pi, \delta_A, \delta_B \bullet \pi \in \mathcal{X} \land A(\delta\langle\pi\rangle, \delta_A) \land B(\delta\langle\bar{\pi}\rangle, \delta_B) \land \delta' = \delta_A + \delta_B \\
c * A &\triangleq \mu X \bullet \mathit{choice}\big((A ; X), \mathit{skip}, \{\iota\langle c\rangle\}\big)
\end{aligned}$$

Fig. 2. UTP semantics for different programming constructs

3.2 Programming Constructs

The semantic definitions of various programming constructs are based on a homogeneous relation between distributions and are listed in Figure 2; we will now proceed to discuss each one.

The failing program abort is represented by the predicate $\|\delta'\| \leq \|\delta\|$, which captures the fact that it is maximally unpredictable, given that it cannot increase distribution weight. Such an increase would describe a program whose probability of termination was higher than that of it starting, and is infeasible.

The miraculous program $\mathit{miracle}$ is defined as $(\delta = \epsilon) \land (\delta' = \epsilon)$: this is a difference in comparison with the standard UTP theory, where it is simply false. This definition coincides with the standard one for most pairs of before- and after-distributions, with the exception of (ϵ, ϵ): this makes sure that $\mathit{miracle}$ is a unit for nondeterministic choice.

Program skip makes no changes and immediately terminates.

Assignment remaps the distribution as has already been discussed in §3.1.

Sequential composition is characterised by the existence of a “mid-point” distribution that is the outcome of the first program, and is then fed into the second. It should be noted at this juncture that we are quantifying over function quantities, such as δ or π — this makes our logic at least second-order, even if the state spaces are finite (the range $[0, 1]$ is not).

The choice operator is probably the strangest-looking element of the list: it takes a weighting distribution π, uses it with its complementary distribution $\bar{\pi} = \iota - \pi$) to weigh the distributions resulting from the left- and right-hand side respectively, and existentially quantifies it over the set of distributions $\mathcal{X} \subseteq \mathcal{D}_w$. We have termed this operator as the *generic choice* as it can emulate the behaviour of all standard choices (and more):

- for $\mathcal{X} = \{\iota\langle c\rangle\}$ we have conditional choice:

$$\begin{aligned}
A \lhd c \rhd B &= \mathit{choice}\big(A, B, \{\iota\langle c\rangle\}\big) \\
&= \exists \delta_A, \delta_B \bullet A(\delta\langle c\rangle, \delta_A) \land B(\delta\langle\neg c\rangle, \delta_B) \land \delta' = \delta_A + \delta_B
\end{aligned}$$

- for $\mathcal{X} = \{p \cdot \iota\}$ we have probabilistic choice:

$$\begin{aligned}
A \;{}_{p}\!\oplus B &= \mathit{choice}\big(A, B, \{p \cdot \iota\}\big) \\
&= \exists \delta_A, \delta_B \bullet A\big(p \cdot \delta, \delta_A\big) \land B\big((1 - p) \cdot \delta, \delta_B\big) \land \delta' = \delta_A + \delta_B
\end{aligned}$$

- for $\mathcal{X} = \mathcal{D}_w$ we have non-deterministic choice:

$$\begin{aligned} A \sqcap B &= \mathit{choice}(A, B, \mathcal{D}_w) \\ &= \exists \pi, \delta_A, \delta_B \bullet A(\delta\langle\pi\rangle, \delta_A) \wedge B(\delta\langle\bar{\pi}\rangle, \delta_B) \wedge \delta' = \delta_A + \delta_B \end{aligned}$$

The usual notations for conditional, probabilistic and non-deterministic choice will be used as syntactic sugar in the remainder of this document. Program abort is a zero for non-deterministic choice, as entering $\|\delta'\| \leq \|\delta\|$ for B in the definition, results in

$$\exists \pi, \delta_A, \delta_B \bullet A(\delta\langle\pi\rangle, \delta_A) \wedge \|\delta_B\| \leq \|\delta\langle\bar{\pi}\rangle\| \wedge \delta' = \delta_A + \delta_B$$

which, after the one-point rule with $\delta_B = \delta' - \delta_A$ reduces to

$$\exists \pi, \delta_A \bullet A(\delta\langle\pi\rangle, \delta_A) \wedge \|\delta' = \delta_A\| \leq \|\delta\langle\bar{\pi}\rangle\|$$

We can take $\pi = \epsilon$ as a witness, which forces $\delta_A = \epsilon$ (by healthiness condition Dist1, see Section 3.3) and we obtain

$$(A(\epsilon, \epsilon) \wedge \|\delta' - \epsilon\| \leq \|\delta\|) \vee \exists \pi, \delta_A \bullet \ldots$$

A consequence of Dist1 is that $A(\epsilon, \epsilon)$ is always true for healthy A so this reduces to $\|\delta'\| \leq \|\delta\|$ in a disjunction with a predicate that it subsumes, and hence equivalent to abort.

As commonly seen in UTP, disjunction of two programs is a kind of choice (usually non-deterministic, in other theories), which here can be defined using generic choice:

$$A \vee B = \mathit{choice}(A, B, \{\epsilon, \iota\}).$$

Disjunction is the usual semantics for non-deterministic choice, but here we see that non-deterministic choice has a richer behaviour as it exhibits more variability. Nevertheless with the appropriate definition of refinement we can introduce a concept of equivalence (*i.e.* two programs mutually refine each other), that restores the equivalence between disjunction and non-deterministic choice. [BB11]

Using the customary notation for conditional choice enlightens the definition of while-loops, which can be rewritten in a more familiar fashion as:

$$c * A \mathrel{\hat{=}} \mu X \bullet (A; X) \lhd c \rhd \mathit{skip}$$

They are characterized as fixpoints of the appropriate functional, with respect to the ordering defined by the refinement relation, details of which can be found in [MM04, BB11] and are beyond the scope of this paper.

These are the most significant elements and constructs that characterise our framework: this has been a presentation from a fairly high level, and it should have provided the reader with a working knowledge of the framework; a formal and rigorous definition of the elements presented so far is beyond the scope of this paper and can be found in [BB11], along with some soundness proofs.

3.3 Healthiness Conditions

Before moving further on, we are going to list quickly the healthiness conditions that characterise this framework.

The first one (*feasibility*, Dist1) assures that for any program $\mathcal{P}(\delta, \delta')$ the probability of termination cannot be greater than that of having started:

$$\|\delta'\| \leq \|\delta\|$$

Another healthiness condition (*monotonicity*, Dist2), states that, for any deterministic program $\mathcal{P}$, increasing δ implies that the resulting δ' increases as well:

$$\mathcal{P}(\delta_1, \delta_1') \wedge \mathcal{P}(\delta_2, \delta_2') \wedge \delta_2 > \delta_1 \Rightarrow \delta_2' \geq \delta_1'$$

A third healthiness conditions is that multiplication by a (not too large and non-negative[3]) constant distributes through commands (*scaling*, Dist3):

$$\forall a \in \mathbb{R}^+ \wedge \|a \cdot \delta\| \leq 1 \bullet \mathcal{P}(\delta, \delta') \Leftrightarrow \mathcal{P}(a \cdot \delta, a \cdot \delta')$$

Finally the purely random non-deterministic model adopted in the distributional framework yields a fourth healthiness condition Dist4 (*convexity*):

$$(\mathcal{P}_1 \sqcap \mathcal{P}_2)(\delta, \delta') \Rightarrow \delta' \geq \min\big(\mathcal{P}_1(\delta) \cup \mathcal{P}_2(\delta)\big)$$

Here $\mathcal{P}_1(\delta)$ denotes the set of all δ' that satisfy $\mathcal{P}_1(\delta, \delta')$.

This poses restrictions on the space of possible program images, which is strictly a subset of $\wp\mathcal{D}$: this is analogous to the set $\mathbb{H}S$ from [MM04].

4 A Probabilistic Theory of Designs

A distinguishing characteristic of designs is the use of the auxiliary variables ok and ok'. They are not sufficient in a probabilistic setting, as we need to be able to express quantitative information about the program also in terms of it having started or finished. We argue that this information is embedded in the distributions used to express programming constructs.

In fact the variable δ records implicitly if the program has started, as for each state σ it gives a precise probability that the program is in that initial state.

If δ is a full distribution (*i.e.* $\|\delta\| = 1$), then the program has started with probability 1: in some sense we can translate the statement $\mathit{ok} = \mathit{true}$ with the statement $\|\delta\| = 1$. Conversely a program for which $\delta = \epsilon$ has not started. Obviously there are all situations in between, where the fact of δ being a subdistribution accounts for the program having started with probability $\|\delta\| < 1$.

Similarly if δ' is a full distribution, then the program terminates with probability 1: coherently we can translate the statement $\mathit{ok}' = \mathit{true}$ with the statement $\|\delta'\| = 1$. In general the weight of δ' is the probability of termination: if the program reaches an after-distribution whose weight is strictly less than 1, then termination is not guaranteed (and in particular if $\delta' = \epsilon$ it is certain that it will not terminate).

[3] Mathematically the relation holds also if this is not met, but in that case $a \cdot \delta$ is not a probability distribution.

4.1 From Standard Designs to Probabilistic Designs

Given a standard design $\mathit{Pre} \vdash \mathit{Post}$ we can easily derive the corresponding probabilistic design by using the observation above:

$$\begin{aligned}\mathit{Pre} \vdash \mathit{Post} &\equiv \mathit{ok} \wedge \mathit{Pre} \Rightarrow \mathit{ok}' \wedge \mathit{Post}\\ &\equiv \|\delta\| = 1 \wedge \mathit{Pre} \Rightarrow \|\delta'\| = 1 \wedge \mathit{Post}\\ &\equiv \|\delta\langle\mathit{Pre}\rangle\| = 1 \Rightarrow \|\delta'\langle\mathit{Post}\rangle\| = 1\end{aligned}$$

This expression tells us that we have a valid design if whenever the before-distribution δ is a full distribution which is null everywhere Pre is not satisfied (and therefore $\delta = \delta\langle\mathit{Pre}\rangle$), then the resulting after-distribution δ' is a full distribution which is null everywhere Post is not satisfied (and therefore $\delta' = \delta'\langle\mathit{Post}\rangle$).

We can easily redefine assignment, in the same style as it has been redefined to make it a valid construct according to the theory of designs:

$$\begin{aligned}\underline{v} := \underline{e} \;&\widehat{=}\; \mathit{true} \vdash \delta' = \delta\{\!|\underline{e}/\underline{v}|\!\}\\ &\equiv \mathit{ok} \wedge \mathit{true} \Rightarrow \mathit{ok}' \wedge \delta\{\!|\underline{e}/\underline{v}|\!\}\\ &\equiv \|\delta\| = 1 \Rightarrow \|\delta'\| = 1 \wedge \delta' = \delta\{\!|\underline{e}/\underline{v}|\!\}\end{aligned}$$

This states that an assignment is a valid design only if the expression e is defined everywhere in the state space: in fact undefinedness of e causes $\delta\{\!|\underline{e}/\underline{v}|\!\}$ to be a sub-distribution and therefore $\underline{v} := e$ reduces to false.

We can redefine skip in a similar way:

$$\begin{aligned}\mathit{skip} \;&\widehat{=}\; \mathit{true} \vdash \delta' = \delta\\ &\equiv \mathit{ok} \wedge \mathit{true} \Rightarrow \mathit{ok}' \wedge \delta\\ &\equiv \|\delta\| = 1 \Rightarrow \|\delta'\| = 1 \wedge \delta' = \delta\\ &\equiv \|\delta\| = 1 \Rightarrow \delta' = \delta\end{aligned}$$

This new version of skip states that the after-distribution is the same as the before-distribution (and therefore it does not alter the weight, so this can be left implicit), but as any other design it reduces to true if δ is not a full distribution.

The bottom of the lattice is abort, which is again true as in the standard theory:

$$\begin{aligned}\mathit{abort} \;&\widehat{=}\; \mathit{false} \vdash \mathit{false}\\ &\equiv \mathit{ok} \wedge \mathit{false} \Rightarrow \mathit{ok}' \wedge \mathit{false}\\ &\equiv \mathit{false} \Rightarrow \mathit{false}\\ &\equiv \mathit{true}\\ &\equiv \mathit{false} \Rightarrow \mathit{true}\\ &\equiv \mathit{ok} \wedge \mathit{false} \Rightarrow \mathit{ok}' \wedge \mathit{true}\\ &\equiv \mathit{false} \vdash \mathit{true}\end{aligned}$$

The standard definition of the construct chaos is

$$\begin{aligned} \mathit{chaos} &\mathrel{\widehat=} \mathit{true} \vdash \mathit{true} \\ &\equiv \mathit{ok} \wedge \mathit{true} \Rightarrow \mathit{ok}' \wedge \mathit{true} \\ &\equiv \mathit{ok} \Rightarrow \mathit{ok}' \\ &\equiv \|\delta\| = 1 \Rightarrow \|\delta'\| = 1 \end{aligned}$$

This is a program that guarantees termination, but in an unspecified state. It is equivalent to:

$$\mathit{chaos} \equiv \mathit{true} \vdash \mathit{abort}_R\,,$$

where the subscript R indicates that we are talking of the relational version of abort, from Figure 2.

The top of the lattice is $\mathit{miracle}$:

$$\begin{aligned} \mathit{miracle} &\mathrel{\widehat=} \mathit{true} \vdash \mathit{false} \\ &\equiv \mathit{ok} \wedge \mathit{true} \Rightarrow \mathit{ok}' \wedge \mathit{false} \\ &\equiv \mathit{ok} \Rightarrow \mathit{false} \\ &\equiv \neg \mathit{ok} \\ &\equiv \neg(\|\delta\| = 1) \\ &\equiv \|\delta\| < 1 \end{aligned}$$

This is equivalent to

$$\mathit{miracle} \equiv \mathit{true} \vdash \mathit{miracle}_R\,.$$

Healthiness Conditions These new definitions relying on the distributional framework satisfy the healthiness conditions H1–H4 as well (§2.2).

We can in fact prove that the following laws hold:

- left unit law:

$$\begin{aligned} \mathit{skip}; \mathit{Pre} \vdash \mathit{Post} &\equiv (\|\delta\| = 1 \Rightarrow \delta' = \delta)\,; (\|\delta\langle \mathit{Pre}\rangle\| = 1 \Rightarrow \|\delta'\langle \mathit{Post}\rangle\| = 1) \\ &\equiv \exists \delta_m \bullet (\|\delta\| = 1 \Rightarrow \delta_m = \delta) \;\wedge\; (\|\delta_m\langle \mathit{Pre}\rangle\| = 1 \Rightarrow \|\delta'\langle \mathit{Post}\rangle\| = 1) \\ &\equiv \|\delta\langle \mathit{Pre}\rangle\| = 1 \Rightarrow \|\delta'\langle \mathit{Post}\rangle\| = 1 \\ &\equiv \mathit{Pre} \vdash \mathit{Post} \end{aligned}$$

- right unit law:

$$\begin{aligned} \mathit{Pre} \vdash \mathit{Post}; \mathit{skip} &\equiv (\|\delta\langle \mathit{Pre}\rangle\| = 1 \Rightarrow \|\delta'\langle \mathit{Post}\rangle\| = 1)\,; (\|\delta\| = 1 \Rightarrow \delta' = \delta) \\ &\equiv \exists \delta_m \bullet (\|\delta\langle \mathit{Pre}\rangle\| = 1 \Rightarrow \|\delta_m\langle \mathit{Post}\rangle\| = 1) \;\wedge\; (\|\delta_m\| = 1 \Rightarrow \delta' = \delta_m) \\ &\equiv \|\delta\langle \mathit{Pre}\rangle\| = 1 \Rightarrow \|\delta'\langle \mathit{Post}\rangle\| = 1 \\ &\equiv \mathit{Pre} \vdash \mathit{Post} \end{aligned}$$

- left zero law:

$$\begin{aligned} \mathit{true}; \mathcal{P}re \vdash \mathcal{P}ost &\equiv \mathit{true}; (\|\delta\langle \mathcal{P}re\rangle\| = 1 \Rightarrow \|\delta'\langle \mathcal{P}ost\rangle\| = 1) \\ &\equiv \exists \delta_m \bullet \mathit{true} \ \wedge \ (\|\delta_m\langle \mathcal{P}re\rangle\| = 1 \Rightarrow \|\delta'\langle \mathcal{P}ost\rangle\| = 1) \\ &\equiv \exists \delta_m \bullet \|\delta_m\langle \mathcal{P}re\rangle\| = 1 \Rightarrow \|\delta'\langle \mathcal{P}ost\rangle\| = 1 \\ &\equiv \mathit{true} \end{aligned}$$

- right zero law:

$$\begin{aligned} \mathcal{P}re \vdash \mathcal{P}ost; \mathit{true} &\equiv (\|\delta\langle \mathcal{P}re\rangle\| = 1 \Rightarrow \|\delta'\langle \mathcal{P}ost\rangle\| = 1); \mathit{true} \\ &\equiv \exists \delta_m \bullet (\|\delta\langle \mathcal{P}re\rangle\| = 1 \Rightarrow \|\delta_m\langle \mathcal{P}ost\rangle\| = 1) \ \wedge \ \mathit{true} \\ &\equiv \exists \delta_m \bullet \|\delta\langle \mathcal{P}re\rangle\| = 1 \Rightarrow \|\delta_m\langle \mathcal{P}ost\rangle\| = 1 \\ &\equiv \mathit{true} \end{aligned}$$

4.2 Recasting Total Correctness

The reason that led to the standard theory of designs was that programs fail to satisfy the left zero law in the relational theory.

In the distributional framework programming constructs do satisfy this law, as for any programming construct $\mathcal{P}$ other than *miracle* it is never the case that $\delta \notin \mathit{fv}(\mathcal{P})$.

For this reason we have:

$$\begin{aligned} \mathit{true}; \mathcal{P}(\delta, \delta') &\equiv \exists \delta_m \bullet \mathit{true} \ \wedge \ \mathcal{P}(\delta_m, \delta') \\ &\equiv \exists \delta_m \bullet \mathcal{P}(\delta_m, \delta') \\ &\equiv \mathit{true} \end{aligned}$$

Similarly the right zero law is satisfied as well, along with the left and right unit laws: healthiness conditions equivalent to H1–H4 hold here as well.

Following this observation it appears that restricting the reasoning to programs with guaranteed termination is somehow limiting, as guaranteed termination is not an actual real-world feature of programs: programs must be reasonably reliable, but failure is always a possibility.

The reason for this may be inherent to the fact that programs are run on hardware which is susceptible of failure, as well as being imputable to the way a program is designed (for example the implementation of a probabilistic algorithm where termination is probabilistic as well).

We can fully exploit the potential of the distributional framework towards modelling these situations by removing the constraints on the weights of the before- and after-distributions — so we use the programming constructs in Figure 2 exactly with the semantics presented there.

The role of preconditions and postconditions is that of restricting the range of acceptable before- and after-distributions (and therefore act as restrictions to be applied to δ and δ' respectively) — this allows us to express desirable characteristics of a program in great detail, for example:

- $\mathcal{P} \wedge \|\delta'\| = 1$ requires $\mathcal{P}$ to guarantee termination;
- $\mathcal{P} \wedge \|\delta'\| > 0.95$ requires $\mathcal{P}$ to terminate with at least 95% probability;
- $\mathcal{P} \wedge \|\delta'(\mathcal{P}ost)\| > 0.95$ requires $\mathcal{P}$ to terminate with at least 95% probability in a state satisfying $\mathcal{P}ost$;
- $\mathcal{P}re \Rightarrow \mathcal{P} \wedge \|\delta'(\mathcal{P}ost)\| > 0.95$ requires $\mathcal{P}$ to terminate with at least 95% probability in a state satisfying $\mathcal{P}ost$ whenever it starts in a state satisfying $\mathcal{P}re$;
- $\|\delta(\mathcal{P}re)\| > 0.98 \Rightarrow \mathcal{P} \wedge \|\delta'(\mathcal{P}ost)\| > 0.95$ requires $\mathcal{P}$ to terminate with at least 95% probability in a state satisfying $\mathcal{P}ost$ whenever the probability of $\mathcal{P}re$ being satisfied at the beginning is at least 0.98;
- ...

All healthiness conditions deriving from the distributional framework (Dist1–Dist4) obviously hold here as well; with a small modification we can recast the notion of total correctness by restricting Dist1 to a variant Dist1-TC (which implies Dist1), stating that:

$$\|\delta\| = \|\delta'\|$$

This requires a program to terminate with the same probability p with which it has started:

$$\|\delta\| = p \wedge \mathcal{P}re \Rightarrow \|\delta'\| = p \wedge \mathcal{P}ost$$

4.3 Link with the Standard Model

Standard designs have observations ok, ok', σ, σ'.

$$ok, ok' : \mathbb{B}$$
$$\sigma, \sigma' : \mathcal{S}$$

A standard design is a predicate $\mathcal{P}_S(\sigma, \sigma', ok, ok')$ that states that a program started (if ok is true) in the state σ ends (if ok' is true) in the state σ'.

Probabilistic designs have observations δ, δ'

$$\delta, \delta' : \mathcal{S} \to [0, 1]$$

A probabilistic design is a predicate $\mathcal{P}_D(\delta, \delta')$ stating that a before-distribution δ will be transformed into the after-distribution δ'.

Informally we require the two approaches to yield the same results when we are dealing with point distributions, *i.e.* when the probability of being in a given state is 1.

In order to formalise the link between these two worlds, we define the linking predicate L as:

$$L((\delta, \delta'), (\sigma, \sigma', ok, ok')) \mathrel{\hat{=}} ok \Leftrightarrow (\|\delta'\| = 1) \wedge ok' \Leftrightarrow (\|\delta'\| = 1)$$
$$\wedge \delta = \eta_\sigma \wedge \delta' = \eta_{\sigma'}$$

Here the notation η_σ denotes the point distribution returning 1 for state σ, and 0 elsewhere.

This linking predicate allows us to introduce the following *Galois connections*; first we define the weakest probabilistic design corresponding to a standard design $\mathcal{P}_S$:

$$\forall \sigma, \sigma', \mathit{ok}, \mathit{ok}' \bullet L((\delta, \delta'), (\sigma, \sigma', \mathit{ok}, \mathit{ok}')) \Rightarrow \mathcal{P}_S(\sigma, \sigma', \mathit{ok}, \mathit{ok}')$$

Analogously, the strongest standard design corresponding to a probabilistic design $\mathcal{P}_D$ is:

$$\exists \delta, \delta' \bullet L((\delta, \delta'), (\sigma, \sigma', \mathit{ok}, \mathit{ok}')) \wedge \mathcal{P}_D(\delta, \delta')$$

It is easy to see that all programming constructs from the probabilistic theory that have homologue ones in the standard theory are linked to them, with the restriction of operating only on point distributions, otherwise they reduce to abort.

Weakening the Link. This linking predicate is a bit too strong, as it maps many interesting program constructs to the aborting program: an example is that of generic choice, which has no homologue in the standard theory. Ideally a better option would be to relax some constraints and to map generic choice to non-deterministic choice rather than to abort.

In other words we are aiming at a link that loses all probabilistic information about the possible after-states and flattens it to a mere list of them.

This is not straightforward, as the linking predicate L in some sense verifies consistency of δ with respect to σ, ok and of δ' with respect to σ', ok': when the support[4] of the distribution has more than one element, the relation between δ and a state from its domain is too weak to be useful.

The situation is similar to that of a 3D-space, where dots are characterised by their x, y, z coordinates: a transformation creates a space with coordinates x', y', z', whose relation with the undashed coordinates cannot in general be captured by a relation that mentions only one undashed and one dashed coordinate.

So far we have seen standard designs as relations:

$$\mathcal{P}_S : \mathcal{S} \times \mathbb{B} \to \mathcal{S} \times \mathbb{B}$$

but in order to build a more useful link we turn to this other interpretation:

$$\mathcal{P}_{\wp S} : \mathcal{S} \times \mathbb{B} \to \wp\mathcal{S} \times \mathbb{B}$$

which maps a state to what we may term its program image $\mathcal{P}(\sigma)$ (as it is a similar concept to that of program image introduced in §3), which contains all of the possible after-states reachable from a given before-state:

$$\mathcal{P}(\sigma) = \{\sigma' \mid \mathcal{P}_S(\sigma, \sigma')\}$$

All deterministic standard constructs map a state to a singleton set, whereas non-deterministic choice maps it to larger sets.

[4] We remind the reader that the support of a function is the set of points where the function is not zero-valued: $\mathrm{supp}(\delta) \mathrel{\hat{=}} \mathrm{dom}(\delta) \setminus \ker(\delta)$.

The interpretation of the predicate $\mathcal{P}_{\wp S}(\sigma, \alpha', ok, ok')$ is therefore that $\mathcal{P}$ has started (if ok is true) in the state σ and has ended (if ok' is true) in a state $\sigma' \in \alpha'$:

$$\mathcal{P}_{\wp S}(\sigma, \alpha', ok, ok') \equiv \bigvee_{\sigma' \in \alpha'} \mathcal{P}_S(\sigma, \sigma', ok, ok')$$

With this in mind we can define the following linking predicate:

$$\begin{aligned} L_{\wp}((\delta, \delta'), (\sigma, \alpha', ok, ok')) \mathrel{\hat{=}} ok \Leftrightarrow (\|\delta'\| = 1) \wedge ok' \Leftrightarrow (\|\delta'\| = 1) \\ \wedge \delta = \eta_\sigma \wedge \operatorname{supp}(\delta') = \alpha' \end{aligned}$$

We can state the variants of the Galois connections above as:

$$\forall \sigma, \alpha', ok, ok' \bullet L_{\wp}((\delta, \delta'), (\sigma, \alpha', ok, ok')) \Rightarrow \mathcal{P}_{\wp S}(\sigma, \alpha', ok, ok')$$
$$\exists \delta, \delta' \bullet L_{\wp}((\delta, \delta'), (\sigma, \alpha', ok, ok')) \wedge \mathcal{P}_D(\delta, \delta')$$

5 An Example: Interaction of Probabilistic and Non-deterministic Choice

This brief classical example is meant to show the interaction and the difference between probabilistic and non-deterministic choice: we will use this to show the effect of projecting the probabilistic design on the space of standard designs.

Let us take these two simple programs:

$$\begin{aligned} A &\mathrel{\hat{=}} x := 0 \sqcap x := 1 \ ; \ y := 0 \ {}_{\frac{1}{2}}\!\oplus y := 1 \\ B &\mathrel{\hat{=}} x := 0 \ {}_{\frac{1}{2}}\!\oplus x := 1 \ ; \ y := 0 \sqcap y := 1 \end{aligned}$$

In Figure 3 we have worked out parametric expression for the final distribution for each program — they are parametric in the weighting distribution π which accounts for the non-deterministic choice performed in both programs:

$$\begin{aligned} \delta'_A(\pi) &\mathrel{\hat{=}} 1/2 \cdot \big(\delta\langle\pi\rangle\{\!|0/x|\!\}\{\!|0/y|\!\} + \delta\langle\bar{\pi}\rangle\{\!|1/x|\!\}\{\!|0/y|\!\} + \delta\langle\pi\rangle\{\!|0/x|\!\}\{\!|1/y|\!\} + \delta\langle\bar{\pi}\rangle\{\!|1/x|\!\}\{\!|1/y|\!\}\big) \\ \delta'_B(\pi) &\mathrel{\hat{=}} 1/2 \cdot \big(\delta\{\!|0/x|\!\}\langle\pi\rangle\{\!|0/y|\!\} + \delta\{\!|1/x|\!\}\langle\pi\rangle\{\!|0/y|\!\} + \delta\{\!|0/x|\!\}\langle\bar{\pi}\rangle\{\!|1/y|\!\} + \delta\{\!|1/x|\!\}\langle\bar{\pi}\rangle\{\!|1/y|\!\}\big) \end{aligned}$$

The two after-distributions are very similar, but with one crucial difference: the position of $\langle\pi\rangle$, which clearly marks when the non-deterministic choice was made; this is reflected in the different after-distributions reached by each program:

- $\forall \pi \bullet \|\delta'_A(\pi)\langle x = y\rangle\| = 1/2$, *i.e.* regardless of the non-deterministic choice and of the initial distribution program A terminates in a state satisfying the condition $x = y$ with probability $1/2$, whereas in program B we cannot remove the dependence on π so if we turn to a worst-case analysis (in the non-deterministic choice the left-hand side $y := 0$ is picked whenever $x = 1$, *i.e.* $\pi = \iota\langle x = 1\rangle$) we have that $\|\delta'_B(\iota\langle x = 1\rangle)\langle x = y\rangle\| = 0$ and therefore the minimum guaranteed probability that $x = y$ is 0;

$$\begin{aligned}
A \quad &\hat{=} \quad x := 0 \sqcap x := 1 \ ; \ y := 0 \ {}_{\frac{1}{2}}\!\oplus y := 1 \\
&\equiv \quad \exists \pi \bullet \delta' = \delta\langle\pi\rangle\{\!|{}^0\!/_x|\!\} + \delta\langle\bar{\pi}\rangle\{\!|{}^1\!/_x|\!\} \ ; \ \delta' = {}^1\!/_2 \cdot \delta\{\!|{}^0\!/_y|\!\} + {}^1\!/_2 \cdot \delta\{\!|{}^1\!/_y|\!\} \\
&\equiv \quad \exists \pi, \delta_m \bullet \delta_m = \delta\langle\pi\rangle\{\!|{}^0\!/_x|\!\} + \delta\langle\bar{\pi}\rangle\{\!|{}^1\!/_x|\!\} \ \wedge \ \delta' = {}^1\!/_2 \cdot \delta_m\{\!|{}^0\!/_y|\!\} + {}^1\!/_2 \cdot \delta_m\{\!|{}^1\!/_y|\!\} \\
&\equiv \quad \exists \pi \bullet \delta' = {}^1\!/_2 \cdot \big(\delta\langle\pi\rangle\{\!|{}^0\!/_x|\!\} + \delta\langle\bar{\pi}\rangle\{\!|{}^1\!/_x|\!\}\big)\{\!|{}^0\!/_y|\!\} + \\
&\qquad\qquad\qquad + {}^1\!/_2 \cdot \big(\delta\langle\pi\rangle\{\!|{}^0\!/_x|\!\} + \delta\langle\pi\rangle\{\!|{}^1\!/_x|\!\}\big)\{\!|{}^1\!/_y|\!\} \\
&\equiv \quad \exists \pi \bullet \delta' = \delta'_A(\pi) \ \wedge \ \delta'_A(\pi) \mathrel{\hat{=}} {}^1\!/_2 \cdot \delta\langle\pi\rangle\{\!|{}^0\!/_x|\!\}\{\!|{}^0\!/_y|\!\} + {}^1\!/_2 \cdot \delta\langle\bar{\pi}\rangle\{\!|{}^1\!/_x|\!\}\{\!|{}^0\!/_y|\!\} + \\
&\qquad\qquad\qquad + {}^1\!/_2 \cdot \delta\langle\pi\rangle\{\!|{}^0\!/_x|\!\}\{\!|{}^1\!/_y|\!\} + {}^1\!/_2 \cdot \delta\langle\bar{\pi}\rangle\{\!|{}^1\!/_x|\!\}\{\!|{}^1\!/_y|\!\}
\end{aligned}$$

$$\begin{aligned}
B \quad &\hat{=} \quad x := 0 \ {}_{\frac{1}{2}}\!\oplus x := 1 \ ; \ y := 0 \sqcap y := 1 \\
&\equiv \quad \delta' = {}^1\!/_2 \cdot \delta\{\!|{}^0\!/_x|\!\} + {}^1\!/_2 \cdot \delta\{\!|{}^1\!/_x|\!\} \ ; \ \exists \pi \bullet \delta' = \delta\langle\pi\rangle\{\!|{}^0\!/_y|\!\} + \delta\langle\bar{\pi}\rangle\{\!|{}^1\!/_y|\!\} \\
&\equiv \quad \exists \pi, \delta_m \bullet \delta_m = {}^1\!/_2 \cdot \delta\{\!|{}^0\!/_x|\!\} + {}^1\!/_2 \cdot \delta\{\!|{}^1\!/_x|\!\} \ \wedge \ \delta' = \delta_m\langle\pi\rangle\{\!|{}^0\!/_y|\!\} + \delta_m\langle\bar{\pi}\rangle\{\!|{}^1\!/_y|\!\} \\
&\equiv \quad \exists \pi \bullet \delta' = \big({}^1\!/_2 \cdot \delta\{\!|{}^0\!/_x|\!\} + {}^1\!/_2 \cdot \delta\{\!|{}^1\!/_x|\!\}\big)\langle\pi\rangle\{\!|{}^0\!/_y|\!\} + \\
&\qquad\qquad\qquad + \big({}^1\!/_2 \cdot \delta\{\!|{}^0\!/_x|\!\} + {}^1\!/_2 \cdot \delta\{\!|{}^1\!/_x|\!\}\big)\langle\bar{\pi}\rangle\{\!|{}^1\!/_y|\!\} \\
&\equiv \quad \exists \pi \bullet \delta' = \delta'_B(\pi) \ \wedge \ \delta'_B(\pi) \mathrel{\hat{=}} {}^1\!/_2 \cdot \delta\{\!|{}^0\!/_x|\!\}\langle\pi\rangle\{\!|{}^0\!/_y|\!\} + {}^1\!/_2 \cdot \delta\{\!|{}^1\!/_x|\!\}\langle\pi\rangle\{\!|{}^0\!/_y|\!\} + \\
&\qquad\qquad\qquad + {}^1\!/_2 \cdot \delta\{\!|{}^0\!/_x|\!\}\langle\bar{\pi}\rangle\{\!|{}^1\!/_y|\!\} + {}^1\!/_2 \cdot \delta\{\!|{}^1\!/_x|\!\}\langle\bar{\pi}\rangle\{\!|{}^1\!/_y|\!\}
\end{aligned}$$

Fig. 3. Programs A and B

- viceversa for program B we can show that $\forall \pi \bullet \|\delta'_B(\pi)\langle x = 1\rangle\| = {}^1\!/_2$ similarly as above, and so the probability that $x = 1$ after program B is ${}^1\!/_2$, whereas if we take program A we can see that if $\pi = \iota$ (*i.e.* in the non-deterministic choice the left-hand side $x := 0$ is always picked) than we have that $\|\delta'_A(\iota)\langle x = 1\rangle\| = 0$, so the minimum guaranteed probability that $x = 1$ is 0.

We are now going to derive the strongest standard design corresponding to A and B using the linking predicate $L_\wp$:

$$\begin{aligned}
&\exists \delta, \delta' \bullet L_\wp((\delta, \delta'), (\sigma, \alpha', ok, ok')) \wedge A(\delta, \delta') \\
\equiv \quad &\text{Definition of } L_\wp \\
&\exists \delta, \delta' \bullet \big(ok \Leftrightarrow (\|\delta'\| = 1) \wedge ok' \Leftrightarrow (\|\delta'\| = 1) \wedge \delta = \eta_\sigma \wedge \operatorname{supp}(\delta') = \alpha'\big) \wedge A(\delta, \delta') \\
\equiv \quad &A \text{ and } B \text{ return after-distributions with the same support} \\
&\exists \delta, \delta' \bullet \big(ok \Leftrightarrow (\|\delta'\| = 1) \wedge ok' \Leftrightarrow (\|\delta'\| = 1) \wedge \delta = \eta_\sigma \wedge \operatorname{supp}(\delta') = \alpha'\big) \wedge B(\delta, \delta')
\end{aligned}$$

The last line of this derivation clearly shows the effect of the link, which flattens out all probabilistic information and as a result the programs A and B are mapped to the same program in the world of standard designs: such a program records that a choice[5] was made, but there is no discrimination among choices of different kind. Moreover what matters is the set of possible after-states and this is also not affected by altering the order in which the choices are made.

[5] Conditional choice is excluded, as it is not really a choice but rather a different evolution of the program which was determined by the current program state.

6 Conclusion

We have presented a probabilistic theory of designs, which relies on a UTP-style framework based on distributions over the state space.

We have shown that we are able to embed the standard UTP theory by requiring guaranteed termination from all program constructs, and treating them as the aborting program otherwise.

We have later relaxed this constraint to be able to reason about probabilistic programs: the advantages of this richer approach is that it allows us to express in fine detail the desired behaviour of a program, including its probabilistic aspects.

References

[BB11] Bresciani, R., Butterfield, A.: Towards a UTP-style framework to deal with probabilities. Technical Report TCD-CS-2011-09, FMG, Trinity College Dublin, Ireland (August 2011)

[BB12] Bresciani, R., Butterfield, A.: A UTP Semantics of pGCL as a Homogeneous Relation. In: Derrick, J., Gnesi, S., Latella, D., Treharne, H. (eds.) IFM 2012. LNCS, vol. 7321, pp. 191–205. Springer, Heidelberg (2012)

[But10] Butterfield, A. (ed.): UTP 2008. LNCS, vol. 5713. Springer, Heidelberg (2010)

[CS09] Chen, Y., Sanders, J.W.: Unifying Probability with Nondeterminism. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 467–482. Springer, Heidelberg (2009)

[DS06] Dunne, S., Stoddart, B. (eds.): UTP 2006. LNCS, vol. 4010. Springer, Heidelberg (2006)

[Hal50] Halmos, P.R.: Measure Theory. University Series in Higher Mathematics. D. Van Nostrand Company, Inc., Princeton (1950)

[He10] He, J.: A probabilistic bpel-like language. In: Qin [Qin10], pp. 74–100

[Heh84] Hehner, E.C.R.: Predicative programming part i & ii. Commun. ACM 27(2), 134–151 (1984)

[HJ98] Hoare, C.A.R., He, J.: Unifying Theories of Programming. Prentice Hall International Series in Computer Science (1998)

[Hoa85] Hoare, C.A.R.: Programs are predicates. In: Proceedings of a Discussion Meeting of the Royal Society of London on Mathematical Logic and Programming Languages, pp. 141–155. Prentice-Hall, Upper Saddle River (1985)

[HS06] He, J., Sanders, J.W.: Unifying probability. In: Dunne and Stoddart [ds06], pp. 173–199

[MM04] McIver, A., Morgan, C.: Abstraction, Refinement and Proof For Probabilistic Systems. Monographs in Computer Science. Springer (2004)

[Qin10] Qin, S. (ed.): UTP 2010. LNCS, vol. 6445. Springer, Heidelberg (2010)

A Keisli Composition

Assume a semantic model of the form $S \to \mathbb{F}S$ where $\mathbb{F}$ is a type constructor (functor). The question that naturally arises is how to compose such functions, i.e., given $p : S \to \mathbb{F}T$ and $q : T \to \mathbb{F}U$, how do we compose these to get $(p;q)$:

$S \to \mathbb{F}U$? The standard solution for this is Kleisli lifting and composition which involves two functions with the following signatures:

$$\eta_S : S \to \mathbb{F}S \qquad\qquad \_^* : (S \to \mathbb{F}T) \to (\mathbb{F}S \to \mathbb{F}T)$$

that obey the following laws:

$$\eta_S^* = id_{\mathbb{F}S} \qquad p^* \circ \eta_S = p \qquad (q^* \circ p)^* = q^* \circ p^*$$

The intuition behind these is best understood in a diagram:

$$\begin{tikzcd}
\mathbb{F}S \arrow[r, "p^*"] & \mathbb{F}T \arrow[r, "q^*"] & \mathbb{F}U \\
S \arrow[u, "\eta_S"] \arrow[ur, "p"'] & T \arrow[u, "\eta_T"] \arrow[ur, "q"'] & U \arrow[u, "\eta_U"']
\end{tikzcd}$$

The Kleisli composition of p and q is given by $q^* \circ p$, where $\circ$ denotes regular function composition.

In this paper $\mathbb{F}S = \mathbb{C}(S \to [0,1])$, and we do not use the full lifting (which results in $\mathbb{C}(S \to [0,1]) \to \mathbb{C}(S \to [0,1]))$, but instead lift partway to get $((S \to [0,1]) \to \mathbb{C}(S \to [0,1]))$. This "partway" lifting is one of the stages in giving an explicit definition of the full lifting.