Deriving Probabilistic Semantics Via the 'Weakest Completion'

He Jifeng[1,⋆,⋆⋆], Carroll Morgan[2], and Annabelle McIver[3]

[1] International Institute for Software Technology, United Nations University, Macau
[2] Department of Computer Science and Engineering, The University of New South Wales, Sydney, Australia
[3] Department of Computing, Macquarie University, Sydney, Australia

1 Introduction

A theory of programming is intended to aid the construction of programs that meet their specifications; for such a theory to be useful it should capture (only) the essential aspects of the program's behaviour, that is only those aspects which one wishes to observe. And it should do so in a mathematically elegant – hence tractable – way.

For conventional imperative sequential programs there are these days two principal theories: the relational[1] and the predicate transformer. Because they have been reasonably successful, each has been used many times as a starting point for extensions: well-known examples include treating *timing* and allowing *concurrency*. In each case the extension somehow elaborates the model on which it is based.

In this paper we investigate two aspects of such semantic extensions, one general and one specific. The specific is that we concentrate on adding *probabilistic* behaviour to our programs. The more important general point however is to extend the theories in some sense "automatically" as far as possible. That is, we wish to make only a few explicit assumptions and then to "generate" the rest by following principles. We illustrate such principles with two case studies: the relational (Sec. 3) and predicate-transformer (Sec. 4) approaches to imperative programming.

There are two stages. In the first we extend the "base type" of the model – the states, for relational, and the predicates for transformers – by adding probabilistic information, and as part of that we give an explicit link between the original type and its probabilistic extension (*i.e.* either an injection from the former to the latter, or a retraction in the opposite direction). From that link between the base and the extended type we then attempt to induce automatically an embedding of *programs* over the base type: the technique is to consider the "weakest completion" of a sub-commuting diagram.

⋆ On leave from the SEI of East China Normal University.

⋆⋆ Partly supported by Research Grant 02104 of MoE and 973 project 2002CB312001 of MoST of China.

[1] We take relational to include for example the "programs are predicates" [9, 10] view.

The second stage is to examine the image, in the extended space of programs, of the original (probability-free) model and to determine what algebraic characteristics define it. Such characteristics, often called "healthiness conditions", both reflect the behaviour of "real" programs and allow us to formulate and prove further algebraic laws that are of practical use in program derivation.

In Sec. 3 we use the weakest completion to derive a semantic definition of a probabilistic programming language from the relational semantics [12, 13] of the "standard", that is non-probabilistic, guarded command language; we relate computations of probabilistic programs to computations of imperative programs. In Sec. 4 we show how to use the weakest completion again, this time to generate probabilistic transformer semantics, our second example, for which our staring point is standard predicate-transformer semantics [3].

2 Probabilistic Program Syntax

The language examined in this paper extends the guarded command language [3] by including the probabilistic choice operator $P\ {}_r\!\oplus Q$ which chooses between programs P and Q with probabilities r and $1-r$ respectively. The abstract syntax of the programming language is given below.

$P ::= \bot$	primitive aborting program
$\quad II$	'skip' program
$\quad x{:=}\ e$	assignment
$\quad P \lhd b \rhd P$	conditional
$\quad P \sqcap P$	demonic choice
$\quad P\ {}_r\!\oplus P$	probabilistic choice
$\quad P; P$	sequential composition
$\quad (\mu X \bullet P(X))$	recursion

For ease of exposition we have included an explicit demonic choice operator $\sqcap$ and used a conventional 'if-then-else' conditional: that gives a convenient separation of the two concepts, combined in Dijkstra's general **if** $\cdots$ **fi**.

3 Relational Semantics

Our first example uses weakest completion to extend a relational semantics for the standard language (Sec. 2 without ${}_r\!\oplus$) to a semantics for the probabilistic language (Sec. 2 with its ${}_r\!\oplus$). We use Hoare's 'design' notation [12] to describe relations.

3.1 Standard Semantics

A standard sequential program starts its execution in an initial state, and terminates (if it ever does) in one of a set of final states. We give the relational meaning of a program by a pair of predicates p, R, *design*, with this syntax and interpretation:

$$p(s) \vdash R(s,s') \quad =_{\mathrm{df}} \quad (ok \wedge p) \Rightarrow (ok' \wedge R)$$

where

- ok records the observation that the program has been properly started,
- ok' records the observation that the program has terminated normally (without 'error messages'), and
- s and s' respectively denote the initial and final states of the program, mappings from the set of program variables to their values.

Thus if the program starts in an initial state satisfying the *precondition* p, it will terminate in a final state satisfying the *postcondition* R.

The effect of the 'design' notation $\cdots \vdash \cdots$ is thus to adjoin a Boolean ok to the state space, for the description of proper termination: if the previous command has terminated (ok in the antecedent) and the precondition p holds of the passed-on state, then this command will establish relation R between s and s', and will itself terminate too (ok' in the consequent).

The approach used in the refinement calculus and in VDM [12, 13, 21] for example gives semantics to programs by associating them with designs according to this scheme:

$$\begin{aligned}
\bot &=_{\mathrm{df}} \mathbf{false} \vdash \mathbf{true} \\
I\!I &=_{\mathrm{df}} \mathbf{true} \vdash (s' = s) \\
x{:=}\ e &=_{\mathrm{df}} \mathbf{true} \vdash (s' = s[e/x]) \\
P \sqcap Q &=_{\mathrm{df}} P \vee Q \\
P \lhd b \rhd Q &=_{\mathrm{df}} (b \wedge P) \vee (\neg b \wedge Q) \\
P; Q &=_{\mathrm{df}} P \circ Q \\
(\mu X \bullet P(X)) &=_{\mathrm{df}} \bigvee \{Q \mid \forall ok, s, ok', s' \bullet (Q \Rightarrow P(Q))\}
\end{aligned}$$

where $\circ$ stands for the (relational) composition of designs defined

$$P \circ Q \quad =_{\mathrm{df}} \ \exists\ \hat{ok}, \hat{s} \bullet P(\hat{ok}, \hat{s}/ok', s') \wedge Q(\hat{ok}, \hat{s}/ok, s)\ ,$$

and $s[e/x]$ denotes the new state obtained by updating s at x with e in the usual way.

We mention here some logical properties of designs that will be useful in later calculations; they are proved elsewhere [12].

Theorem 3.1

1. $(p0 \vdash R0) \vee (p1 \vdash R1) \quad = \quad (p0 \wedge p1) \vdash (R0 \vee R1)$
2. $(p0 \vdash R0) \wedge (p1 \vdash R1) \quad = \quad (p0 \vee p1) \vdash (p0 \Rightarrow R0) \wedge (p1 \Rightarrow R1)$
3. $(p0 \vdash R0) \circ (p1 \vdash R1) \quad = \quad (p0 \wedge \neg(R0 \circ \neg p1)) \vdash (R0 \circ R1)$
4. $(p0 \vdash R0) \lhd b \rhd (p1 \vdash R1) \quad = \quad (p0 \lhd b \rhd p1) \vdash (R0 \lhd b \rhd R1)$

□

A program *refines* another if it terminates more often and behaves less nondeterministically than the other. We write $\sqsubseteq$ for "is refined by"; for predicates[2] P and Q it is defined by

$$P \sqsubseteq Q \quad =_{\mathrm{df}} \quad \forall\ s, s', ok, ok' \bullet (Q \Rightarrow P)\ .$$

Theorem 3.2 (Top and Bottom Designs). For all designs D,

$$\bot \quad = \quad (\mathbf{false} \vdash \mathbf{true}) \quad \sqsubseteq \quad D \quad \sqsubseteq \quad (\mathbf{true} \vdash \mathbf{false}) \quad = \top\ .$$

$\square$

3.2 Probabilistic Semantics

We extend our standard states to probabilistic states by replacing the final state s' with a final *distribution* which we call $prob'$. Maintaining the structure of designs in other respects, we say that a probabilistic program P can be identified as a design with ok, s, ok' and $prob'$ as free variables.

Definition 3.3 (Probabilistic Distributions and Designs). Let S be a state space. The set of distributions over S is the set of total functions from S into the closed interval of reals $[0, 1]$,

$$PROB \quad =_{\mathrm{df}} \quad S \rightarrow [0, 1]$$

We insist further that for any member $prob$ of $PROB$ the probabilities must sum to 1:

$$\Sigma_{s \in S}\ prob(s) \ = \ 1\ .$$

For any subset X of S we define

$$prob(X) \quad =_{\mathrm{df}} \quad \Sigma_{s \in X}\, prob(s)\ .$$

We use $\mathcal{P}$ to denote the set of *probabilistic designs* $p(s) \vdash R(s, prob')$.

We begin our extension by postulating a retraction from probabilistic to standard states; from that (below) we will induce an embedding of programs (designs).

The difference between the standard and the probabilistic semantics is that the former tells us which final states are or are not possible, whereas the latter tells us the probability with which they may occur. Thus to relate the latter to the former we take the view that a final state is possible iff it has positive probability of occurrence:

[2] Although strictly speaking we are concerned only with the predicates defined in our design notation, this definition applies for all predicates provided the right-hand universal quantification is understood to be over all free variables. We use that in Definition 3.5 below.

Definition 3.4 (Retraction of States). The design ρ relates a probabilistic state *prob* to a set of standard states s' according to the definition

$$\rho(ok, prob, ok', s') \quad =_{\mathrm{df}} \quad \mathbf{true} \vdash prob(s') > 0\ .$$

Note that we have expressed the retraction — though not properly a computation — in the design notation as well, complete with its implicit *ok* variables; that allows us to use design composition in the construction of our sub-commuting diagram below, from which we extract the embedding of designs.

Now, based on our injection ρ, we seek an embedding of designs:

Definition 3.5 (Embedding of Programs). For any standard design D, we identify its embedding $\mathcal{K}(D)$, which will be a probabilistic design, as the *weakest solution* in X of the equation

$$X \circ \rho \quad = \quad D\ .$$

Thus define

$$\mathcal{K}(D) \ =_{\mathrm{df}} \ D/\rho\ ,$$

where the notation D/ρ represents the *weakest pre-specification*[11] of D through ρ:

$$(X \circ \rho) \sqsupseteq D \qquad \textbf{iff} \qquad X \sqsupseteq D/\rho, \qquad \text{for all predicates } X\ .$$

The following theorem shows that $\mathcal{K}(D)$ is indeed a probabilistic design.

Theorem 3.6.

$$\mathcal{K}(p(s) \vdash R(s,s')) \quad = \quad p(s) \vdash (prob'(R) = 1)\ ,$$

where $prob'(R)$ abbreviates $prob'(\{t \mid R(s,t)\})$.

Proof:

$$\begin{array}{lll}
 & \mathcal{K}(p \vdash R) & \\
= & p(s) \vdash (R/(prob(s') > 0)) & (p \vdash R)/(\mathbf{true} \vdash Q) = p \vdash (R/Q) \\
= & & X/(prob(s') > 0 = \neg(\neg X \circ (prob(s') > 0)) \\
 & p(s) \vdash prob'(\{t \mid R(s,t)\}) = 1\ . &
\end{array}$$

□

Corollary 3.7 (Inverse of $\mathcal{K}$). For all designs D we have

$$\mathcal{K}(D) \circ \rho \quad = \quad D\ .$$

Proof: From Theorems 3.1(3) and 3.6. □

3.3 Preservation of Program Structure

We now examine the distribution of $\mathcal{K}$ through the program constructors of our language. First we note the effect of $\mathcal{K}$ on primitive designs.

Theorem 3.8 (Primitive Probabilistic Designs).

1. $\mathcal{K}(\bot) \quad = \quad \bot.$
2. $\mathcal{K}(x{:=}\ e) \quad = \quad \mathbf{true} \vdash \mathit{prob}'(s[e/x]) = 1.$

□

Now we show that $\mathcal{K}$ distributes through conditional choice.

Theorem 3.9 (Conditional Choice).

$$\mathcal{K}(D0 \lhd b \rhd D1) \quad = \quad \mathcal{K}(D0) \lhd b \rhd \mathcal{K}(D1)\ .$$

Proof: From Theorems 3.1(4) and 3.6. □

We now use $\mathcal{K}$ to discover the appropriate definition for demonic choice between probabilistic designs.

Theorem 3.10 (Demonic Choice).

$$\mathcal{K}(D0 \sqcap D1) \quad = \quad \mathcal{K}(D0) \vee \mathcal{K}(D1) \vee \bigvee_{0<r<1} (\mathcal{K}(D0) \|_{M_r} \mathcal{K}(D1))\ ,$$

where M_r is a 'coupling predicate' in the style of [12] used in this case as

$$\begin{array}{l} (p0 \vdash R0) \|_{M_r} (p1 \vdash R1) \\ =_{\mathrm{df}} ((p0 \wedge p1) \vdash (R0(s, 0.\mathit{prob}') \wedge R1(s, 1.\mathit{prob}')) \circ M_r \end{array}$$

and $M_r \quad =_{\mathrm{df}} \quad \mathbf{true} \vdash \mathit{prob}' = r \times 0.\mathit{prob} + (1-r) \times 1.\mathit{prob}.$

Proof: Let $D0 = p0 \vdash R0$ and $D1 = p1 \vdash R1$, then proceed

RHS

$=$ Def. of $\|_{M_r}$ and Theorem 3.1(3)

$$\begin{array}{l} p0 \vdash \mathit{prob}'(R0) = 1 \\ \vee\ (p1 \vdash \mathit{prob}'(R1) = 1) \\ \vee\ \ \bigvee_{0<r<1} (p0 \wedge p1) \\ \quad \vdash \exists 0.\mathit{prob}, 1.\mathit{prob} \bullet\ 0.\mathit{prob}(R0) = 1 \\ \qquad\qquad\qquad\qquad \wedge\ 1.\mathit{prob}(R1) = 1 \\ \qquad\qquad\qquad\qquad \wedge\ \mathit{prob}' = r \times 0.\mathit{prob} + (1-r) \times 1.\mathit{prob} \end{array}$$

$\sqsupseteq$ $(r \times 0.\mathit{prob} + (1-r) \times 1.\mathit{prob})(R0 \vee R1) = 1$

$$\begin{array}{l} (p0 \vdash \mathit{prob}'(R0 \vee R1) = 1 \\ \vee\ (p1 \vdash \mathit{prob}'(R0 \vee R1) = 1) \\ \vee\ (p0 \wedge p1) \vdash \mathit{prob}'(R0 \vee R1) = 1 \end{array}$$

$$
\begin{array}{lll}
= & (p0 \wedge p1) \vdash prob'(R0 \vee R1) = 1 & \text{Theorem 3.1(1)} \\
= & \mathcal{K}((p0 \wedge p1) \vdash (R0 \vee R1)) & \text{Theorem 3.6} \\
= & LHS & \text{Theorem 3.1(1)}
\end{array}
$$

$$
\begin{array}{ll}
= & \text{Theorem 3.6 and case analysis} \\
& \quad (p0 \wedge p1) \vdash \ prob'(R0) = 1 \ \vee \ prob'(R1) = 1 \\
& \vee \ \exists \alpha, \beta > 0 \bullet prob'(R0 \vee R1) = 1 \\
& \qquad\qquad\qquad \wedge \ prob'(R0 \wedge \neg R1) = \alpha \\
& \qquad\qquad\qquad \wedge \ prob'(R1 \wedge \neg R0) = \beta
\end{array}
$$

$$
\sqsupseteq \qquad \begin{cases} r = \alpha/(\alpha+\beta) \\ 0.prob(R0 \wedge \neg R1) = \alpha + \beta \\ 0.prob(R0 \wedge R1) = 1 - (\alpha+\beta) \\ 1.prob(R1 \wedge \neg R0) = \alpha + \beta \\ 1.prob(R0 \wedge R1) = 1 - (\alpha+\beta) \end{cases}
$$

$$
\begin{array}{l}
\quad (p0 \wedge p1) \vdash prob'(R0) = 1 \\
\vee \ prob'(R1) = 1 \\
\vee \ \exists r \in (0,1), 0.prob, 1.prob \bullet \\
\quad 0.prob(R0) = 1 \wedge \\
\quad 1.prob(R1) = 1 \wedge \\
\quad prob' = r \times 0.prob + (1-r) \times 1.prob
\end{array}
$$

$$
= \qquad RHS \qquad\qquad \text{Theorem 3.1(1) and Def. } \|_{M_r}
$$

For sequential composition we follow the Kleisli-triple approach to semantics of programming languages [20], introducing a function $\uparrow$ to deal with sequential composition, which maps a probabilistic design taking (ok, s) to $(ok', prob')$ to a ‘lifted’ design taking $(ok, prob)$ to $(ok', prob')$.

Definition 3.11 (Kleisli Lifting).

$$
\begin{array}{ll}
& \uparrow (p \vdash R) \\
=_{\text{df}} & (prob(p) = 1) \\
& \vdash \exists Q \in (S \to PROB) \bullet \\
& \quad \forall s \bullet prob(s) > 0 \Rightarrow R(s, Q(s)) \wedge prob' = \Sigma_{t \in S}(prob(t) \times Q(t))
\end{array}
$$

From the facts that $prob \in PROB$ and that for all $t \in S$ we have $Q(t) \in PROB$ we conclude that

$$\Sigma_{t \in S} prob(t) \times G(t) \ \in PROB \ .$$

Theorem 3.12 (Sequential Composition).

$$\mathcal{K}(D0; D1) \quad = \quad \mathcal{K}(D0) \circ \uparrow \mathcal{K}(D1)$$

Proof:

$$
\begin{array}{ll}
 & RHS \\
= & \text{Theorem 3.6} \\
 & \quad p0(s) \vdash prob'(\{t \mid R0(s,t)\}) = 1 \\
 & \circ\ prob(p1) = 1 \vdash \\
 & \quad \exists Q \bullet \forall t \bullet prob(t) > 0 \Rightarrow \quad Q(t)(\{s' \mid R1(t,s')\}) = 1 \\
 & \qquad\qquad\qquad\qquad\qquad \wedge\ prob' = \Sigma_t(prob(t) \times Q(t))) \\
= & \text{Theorem 3.1(3)} \\
 & \quad p0(s) \wedge \neg(prob'(\{t \mid R0(s,t)\}) = 1 \circ prob(p1) < 1) \\
 & \vdash \exists \rho \bullet \quad \rho(\{t \mid R0(s,t)\}) = 1 \\
 & \qquad\quad \wedge\ prob' = \Sigma_s\{\rho(t) \times Q(t) \mid \rho(t) > 0 \\
 & \qquad\quad \wedge\ Q(t)(\{s' \mid R1(t,s')\}) = 1\} \\
\Rightarrow & \qquad prob'(R0 \circ R1) = \Sigma_t\{\rho(t) * Q(t)(\{s' \mid R1(t,s')\}) \mid R0(s,t) \wedge \rho(t) > 0 \\
 & p0(s) \wedge \neg(R0 \circ p1) \ \vdash \ prob'(R0 \circ R1) = 1 \\
= & LHS \qquad \text{Theorems 3.1(3) and 3.6} \\
\Rightarrow & \begin{cases} f(u,v) =_{\text{df}} prob'(v)/(\#\{t \mid R0(s,t) \wedge R1(t,v)\}) \ R0(s,u) \wedge R1(u,v) \ 0 \\ \rho(u) =_{\text{df}} \Sigma_v f(u,v) \\ Q(u)(v) =_{\text{df}} f(u,v)/\rho(u) \text{ if } \rho(u) > 0 \end{cases} \\
 & \quad p0(s) \wedge \neg(R0 \circ \neg p1) \\
 & \vdash \exists \rho \bullet \rho(\{t \mid R0(s,t)\}) = 1 \wedge \\
 & \quad prob' = \Sigma_s\{\rho(t) * \times Q(t) \mid \rho(t) > 0 \wedge Q(t)(\{s' \mid R1(t,s')\}) = 1\} \\
= & RHS\ .
\end{array}
$$

□

Theorem 3.13 (New Unit).

1. $\uparrow \mathcal{K}(II) \ = \ \mathbf{true} \vdash (prob' = prob)$
2. $\mathcal{K}(II) \circ \uparrow P \ = \ P$

□

Following Theorems 3.6 to 3.12 we conclude by adopting the following definition for the probabilistic programming language semantics.

Definition 3.14 (Probabilistic Programs).

$$
\begin{aligned}
\bot &=_{\text{df}} \mathcal{K}(\mathbf{true}) \\
II &=_{\text{df}} \mathcal{K}(\mathbf{true} \vdash (s' = s)) \\
x := e &=_{\text{df}} \mathcal{K}(\mathbf{true} \vdash (s' = s[e/x]))
\end{aligned}
$$

$$\begin{aligned}
P \lhd b \rhd Q &=_{\mathrm{df}} (b \wedge P) \;\vee\; (\neg b \wedge Q) \\
P \sqcap Q &=_{\mathrm{df}} P \;\vee\; Q \;\vee\; \bigvee_{0<r<1} (P \|_{M_r} Q) \\
P \;{}_{r}\!\oplus Q &=_{\mathrm{df}} P \|_{M_r} Q, \qquad \text{when } 0 < r < 1 \\
P \;{}_{1}\!\oplus Q &=_{\mathrm{df}} P \\
P \;{}_{0}\!\oplus Q &=_{\mathrm{df}} Q \\
P; Q &=_{\mathrm{df}} P \circ \uparrow Q \\
(\mu X \bullet P(X)) &=_{\mathrm{df}} \bigvee \{Q \mid Q \sqsupseteq P(Q)\}
\end{aligned}$$

4 Predicate-Transformer Semantics

4.1 Introduction: Embedding Predicates

In Sec. 3 it was shown that a postulated retraction into the standard state space from the probabilistic state space could, with a small number of reasonable assumptions, be used to induce an embedding of standard programs into a space of probabilistic programs. The principal tool was the 'weakest completion' of a sub-commuting diagram, for which we used the weakest pre- and post-specifications.

In this section we apply the weakest completion to probabilistic transformer (as opposed to relational) semantics, for which our starting point is standard predicate-transformer semantics [3].

As before, we have a standard state space S; and for standard predicate transformer semantics we are interested in predicates over S, which we will take to be members of the powerset $\mathbb{P}S$. Given a predicate $P \subseteq S$ and a state s we can say with certainty whether P holds ($s \in P$) or does not hold ($s \notin P$) at s.

In contrast, a 'probabilistic predicate' A holds at s only with some probability: thus the type of A should be $S \to [0, 1]$, which we write $\mathbb{E}S$. (Note that members of $\mathbb{E}S$ are not distributions over S.)

For example, let S be the three-element state space $\{-1, 0, 1\}$ and consider this probabilistic predicate in $\mathbb{E}S$ over the initial state of a given program:

> executing the program
>
> $$s{:=}\; -s \quad {}_{1/3}\!\oplus \quad s{:=}\; +s$$
>
> will establish the postcondition $s \geq 0$.

In (initial) states $-1, 0, 1$ that probabilistic predicate has value $1/3, 1, 2/3$ respectively. (Note that those values do not sum to 1.)

Standard predicates in $\mathbb{P}S$ are injected into $\mathbb{E}S$ simply by converting them to characteristic functions: writing P^* for that injection of P, we note that P holds at s if the probability that P^* holds at s is 1, and that P does not hold at s if the probability that P^* holds at s is 0.

4.2 Embedding Programs

Based on our injection $()^*$ of predicates, we now seek an embedding of programs: for standard transformer t in $\mathbb{P}S \to \mathbb{P}S$ we want a definition of its embedding t^*, which will be a probabilistic predicate transformer in $\mathbb{E}S \to \mathbb{E}S$. For that we use the techniques of Sec. 3.

First we note that the injection of programs should be related to the injection of predicates by

$$t^*(P^*) \quad = \quad t(P)^* \ , \tag{1}$$

since it should not matter whether we analyse standard programs in their 'native' standard space or in the extended probabilistic space. That tells us immediately the action t^* has on *standard elements* of $\mathbb{E}S$, those that are purely $\{0,1\}$-valued (equivalently are P^* for some P).

More challenging is to deduce reasonable behaviour for t^* on *proper* elements of $\mathbb{E}S$, those that are not standard. Take for example the program t to be $s := -s$, acting (as before) over the space $S = \{-1, 0, 1\}$. From (1) we know

$$t^* \begin{pmatrix} -1 \rightsquigarrow 0 \\ 0 \rightsquigarrow 0 \\ 1 \rightsquigarrow 1 \end{pmatrix} \quad = \quad \begin{pmatrix} -1 \rightsquigarrow 1 \\ 0 \rightsquigarrow 0 \\ 1 \rightsquigarrow 0 \end{pmatrix} \ ;$$

But we do not know the value of say

$$t^* \begin{pmatrix} -1 \rightsquigarrow 1/2 \\ 0 \rightsquigarrow 0 \\ 1 \rightsquigarrow 1 \end{pmatrix} \ .$$

Applying the technique of Sec. 3 directly we would choose t^* to be the least transformer satisfying (1), suggesting the result of applying it to the above should be

$$\begin{pmatrix} -1 \rightsquigarrow 0 \\ 0 \rightsquigarrow 0 \\ 1 \rightsquigarrow 0 \end{pmatrix} \ ,$$

But that 'brute force' technique of taking all proper arguments of t^* to the everywhere-0 expectation is unlikely to be useful: for example we would not even have that t^* was monotonic in general.

We turn therefore to 'healthiness conditions' on $\mathbb{E}S \to \mathbb{E}S$, making explicit which conditions — like monotonicity — we will impose.

4.3 Healthiness Conditions

In the minimalist spirit we impose only one principal condition on the probabilistic transformers: continuity. For any u in $\mathbb{E}S \to \mathbb{E}S$ and any directed subset $\mathcal{A}$ of $\mathbb{E}S$ we require

$$t(\sqcup\mathcal{A}) \quad = \quad \sqcup\{t(A) \mid A \in \mathcal{A}\} \ , \tag{2}$$

where the order $\leq$ over which $\sqcup$ is taken is just the normal $\leq$ relation over $[0,1]$ extended pointwise to $\mathbb{E}S$. (Note that it acts as the injection of implication.)

From (2) we extract two more-specialised conditions: monotonicity, and 'scaling'. Monotonicity follows in the usual way from continuity: and as it is $\leq$-monotonicity, it agrees with the usual $\Rrightarrow$-monotonicity of standard transformers when restricted to their embeddings.

To 'discover' scaling we apply continuity to a c-indexed subset $\mathcal{A}$ of $\mathbb{E}S$, with c varying over reals in $[0,1]$: for arbitrary fixed A in $\mathbb{E}A$ and state s we must have that

$$t(c \times A)(s)$$

varies smoothly, passing through each value between $0 = t(0)(s) = t(0 \times A)(s)$ and $t(A)(s)$, as c itself is varied from 0 to 1. The simplest hypothesis is to suppose that the multiplication by c distributes directly, which is the property we call *scaling*:

for all c in $[0,1]$, probabilistic predicate A in $\mathbb{E}S$ and transformer u in $\mathbb{E}S \to \mathbb{E}S$ we have

$$u(c \times A) \quad = \quad c \times u(A)\ . \tag{3}$$

4.4 Completing the Embedding

We now define t^*, for t in $\mathbb{P}S \to \mathbb{P}S$, to be the least monotonic and scaling transformer in $\mathbb{E}S \to \mathbb{E}S$ that satisfies (1). (It will turn out as a consequence that t^* is continuous as well, so satisfying (2).) This section is devoted to exhibiting a constructive definition of t^*.

Lemma 4.1. Let t be a predicate transformer in $\mathbb{P}S \to \mathbb{P}S$. Then for any expectation A we have

$$t^*(A) \quad = \quad \sqcup_{P \in \mathbb{P}S}\, (\sqcap_P A) \times t(P)^*\ ,$$

where $\sqcap_P A$ denotes the infimum of A over the subset P of S.

Proof: Clearly the right-hand side defines a transformer that is monotonic and scaling (in A); to see that it satisfies (1) we take A to be P^* (renaming the bound P to P'), and reason

$$\begin{array}{lll}
 & \sqcup_{P' \in \mathbb{P}S}\ (\sqcap_{P'} P^*) \times t(P')^* & \\
= & \sqcup_{P' \in \mathbb{P}P}\ (\sqcap_{P'} P^*) \times t(P')^* & \text{if } P' \not\subseteq P \text{ then } \sqcap_{P'} P^* = 0 \\
= & \sqcup_{P' \in \mathbb{P}P}\ t(P')^* & \text{if } P' \subseteq P \text{ then } \sqcap_{P'} P^* = 1 \\
= & t(P)^*\ . & \text{monotonicity of } t
\end{array}$$

Now consider any (other) u in $\mathbb{E}S \to \mathbb{E}S$ satisfying the three conditions; we have

$$\begin{array}{lll}
 & u(A) & \\
\geq & & u \text{ monotonic and scaling}
\end{array}$$

$$
\begin{array}{lll}
 & \sqcup\{c \times u(P^*) \mid c \in [0,1]; P \in \mathbb{P}S; c \times P^* \leq A\} & \\
= & \sqcup\{c \times t(P)^* \mid c \times P^* \leq A\} & \text{Property (1)} \\
= & \sqcup\{c \times t(P)^* \mid c \leq \sqcap_P A\} & \text{rewrite condition} \\
= & \sqcup\{(\sqcap_P A) \times t(P)^* \mid P \in \mathbb{P}S\} & c \times t(P)^* \text{ is monotonic in } c \\
= & \sqcup_{P \in \mathbb{P}S}\ (\sqcap_P A) \times t(P)^*\ , &
\end{array}
$$

which shows the right-hand side indeed satisfies the definition of t^*.

As remarked above, we should have t^* continuous if t itself is. Take directed set $\mathcal{A}$ of expectations, let $\mathbb{F}S$ denote the finite subsets of S, and reason

$$
\begin{array}{lll}
 & t^*(\sqcup\mathcal{A}) & \\
= & \sqcup_{P \in \mathbb{P}S}\ (\sqcap_P(\sqcup\mathcal{A})) \times t(P)^* & \text{Lem. 4.1} \\
\leq & \sqcup_{P \in \mathbb{F}S}\ (\sqcap_P(\sqcup\mathcal{A})) \times t(P)^* & t \text{ continuous} \\
= & \sqcup_{P \in \mathbb{F}S; A \in \mathcal{A}}\ (\sqcap_P A) \times t(P)^* & P \text{ finite, } \mathcal{A} \text{ directed} \\
\leq & \sqcup_{A \in \mathcal{A}} \sqcup_{P \in \mathbb{P}S}\ (\sqcap_P A) \times t(P)^* & \mathbb{F}S \subseteq \mathbb{P}S \\
= & \sqcup_{A \in \mathcal{A}}\ t^*(A)\ , & \text{Lem. 4.1}
\end{array}
$$

with the opposite inequality immediate from monotonicity of t^*.

4.5 Program Structure Is Preserved

The previous section obtained the main result, that an embedded t^* can be induced from the embedding of predicates. Here we verify the technical details of the embedding, in particular (as in Sec. 3.3 earlier) investigating the effect on the elementary program structures (Lem. 4.4 below); for convenience we give a technical lemma that characterises $()^*$-images.

Definition 4.2. A transformer u in $\mathbb{E}S \to \mathbb{E}S$ is said to be *semi-linear* if for all c, c' in $[0,1]$ and A in $\mathbb{E}S$ we have

$$u(c{\times}A \ominus \underline{c'}) \quad = \quad c{\times}u(A) \ominus \underline{c'}\ ,$$

where *truncated subtraction* $a \ominus b$ is defined $(a - b) \sqcup 0$ with lowest syntactic precedence, and $\underline{c'}$ is the constant function in $\mathbb{E}S$ taking the value c' everywhere. □

Lemma 4.3. A continuous transformer u in $\mathbb{E}S \to \mathbb{E}S$ is semi-linear iff it is the image t^* of some standard transformer in $\mathbb{P}S \to \mathbb{P}S$.

Proof: For *if* note that Lem. 4.1 allows $t^*(A)(s)$ to be written

$$(\sqcup P \mid s \in t(P) \bullet \sqcap_P A)\ ,$$

and both $(c\times)$ and $(\ominus\underline{c'})$ distribute through $\sqcap_P$ and $\sqcup$. For *only if* the proof is given elsewhere [23].

We now deal with the elementary program structures.

Lemma 4.4. The following structure-preservation laws hold for program embedding.

1. $\bot$ is the least program: $\bot^*(A) = \underline{0}$.
2. Assignment acts as composition, in the usual way: $(x{:=}\ e)^*(A) = A \circ (\lambda s \bullet s[e/x])$.
3. For state-to-Boolean conditional B we have the usual $(t_1 \lhd B \rhd t_2)^* = t_1^* \lhd B \rhd t_2^*$.
4. Demonic choice, acting as conjunction for standard transformers, becomes *minimum* for probabilistic transformers:
$$(t_1 \Box t_2)^*(A) = t_1^*(A) \sqcap t_2^*(A)\ ;$$
5. Sequential composition remains composition of transformers:
$$(t_1; t_2)^* = t_1^*; t_2^*\ .$$

Proof: Properties 1 and 3 come directly from Lem. 4.1. For Property 2 we argue

$$\begin{array}{lll}
 & (x{:=}\ e)^*(A)(s) & \\
= & \sqcup_{P \in \mathbb{P}S}\ (\sqcap_P A) \times ((x{:=}\ e)(P))^*(s) & \text{Lem. 4.1} \\
= & \sqcup_{P \in \mathbb{P}S}\ (\sqcap_P A) \times P^*(s[x/e]) & s \in (x{:=}\ e)(P) \text{ iff } s[e/x] \in P \\
= & \sqcup_{P \ni s[x/e]} \sqcap_P A & P^*(s[x/e]) \text{ is 0 otherwise} \\
= & A(s[x/e])\ . &
\end{array}$$

Properties 4 and 5 require Lem. 4.3. For the first, we have that t_1^* and t_2^* are semi-linear (Lem. 4.3 *if*); and then it is easy to check that semi-linearity is preserved by $\sqcap$. Thus $t_1^*(A) \sqcap t_2^*(A)$ is semi-linear in A, and so for all A equals $t^*(A)$ for some standard t (Lem. 4.3 *only if*). But since the two sides of Property 4 agree on standard arguments, the two standard transformers of which they are images, that is $t_1 \Box t_2$ (*lhs*) and t (*rhs*), must themselves agree on all (standard) arguments. Hence they are equal.

For Property 5 the argument is similar, relying this time on preservation of semi-linearity by composition.

5 Conclusion

Choosing the 'right' semantic extension can be a tricky business, to some extent a lucky guess.[3] Here we have isolated a principle that can reduce the guesswork.

In our two examples (Sections 3, 4) we postulated an extension of the 'base' type to include the desired new information — in this case probability. Although the base types differ — states on the one hand, and predicates (sets of states)

[3] Consider for example the great variety of bisimulations in demonic process algebras, each one (via the quotient) inducing a slightly different semantic model.

on the other — the subsequent procedure was broadly the same[4] in each case: to induce a corresponding embedding of programs, take the weakest completion of a certain sub-commuting diagram.

We have shown in this paper that the embedding is actually a homomorphism, *i.e.* it distributes over appropriately-defined programming operators. As a result, most of the algebraic laws, which were established in the original semantical framework and were used to capture the properties of the programming operators, remain valid in the enriched model.

The resulting probabilistic models have each been shown independently to be useful in their own right [7, 23]; and in further work [8] it is hoped that the value of the technique will be further demonstrated.

The weakest completion approach has been used for in support of data refinement in VDM and other state-based development methods, where the link is designed to connect abstract data type with its concrete representation. It is also used for data abstraction in model checking by dramatically reducing the size of state space. This paper illustrates another way of using simulation to derive enriched semantics.

References

1. J.C.M. Baeten, J.A. Bergstra and S.A. Smolka. *Axiomatizing Probabilistic Processes ACP with Generative Probabilities.* LNCS 630, (1993).
2. R.-J.R. Back. *A calculus for program derivations*, Acta Informatica 25, 593-624, (1988).
3. E.W. Dijkstra. *A Discipline of Programming*, Prentice-Hall, (1976).
4. R. Fagin, J.Y. Halpern and N. Meggido. *A logic for reasoning about probabilities*, Information and Computation, 78-128, (1990).
5. Michele Giry. *A categorical approach to probability theory*, LCM 915, 68–85, (1975).
6. H. Hansson and B. Jonsson. *A calculus for communicating systems with time and probabilities.* In Proceedings of the 11th IEEE Symposium on Real-time systems (1990).
7. He Jifeng, K. Seidel and A. McIver. *Probabilistic models for the Guarded Command Language* Science of Computer Programming, 28:171–192, (1997).
8. He Jifeng. *Derive Enriched Semantics by Simulation.* To appear in the Proceedings of the Workshop of Refinement and Abstraction, ETL, Osaka, (1999).
9. E.C.R. Hehner. *Predicative Programming: Part 1 and 2.* Communications of the ACM, Vol 27(2): 134–151, (1984).
10. E.C.R. Hehner. *A Practical Theory of Programming.* Springer-Verlag, (1993)
11. C.A.R. Hoare and He Jifeng *Weakest prespecifications.* Fundamenta Informaticae IX, 51–84, 217–252, (1986).
12. C.A.R. Hoare and He Jifeng *Unifying theories of programming*, Prentice-Hall, (1998).
13. C.B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall, (1986).
14. C. Jones and G. Plotkin. *A probabilistic power domain of evaluations*, In Proceedings of 4th IEEE Symposium on Logic in Computer Science, 186-195, Cambridge, Mass., (1989)

[4] For the predicates we imposed 'scaling' additionally.

15. C. Jones. *Probabilistic Nondeterminism*, Doctoral Thesis, Edinburgh University.
16. D. Kozen. *Semantics of probabilistic programming.* Journal of Computer Systems Science 22, 328–350, (1981).
17. D. Kozen. *a probabilistic PDL.* Proceedings of the 15th ACM Symposium on Theory of Computing. (1983)
18. K.G. Larsen and A. Skou. *Bisimulation through probabilistic testing.* Information and Computation 94(1), (1991).
19. F.W. Lawvere. *The category of probabilistic mappings*, Preprint, (1962).
20. E. Moggi. *Notations of computation and monads.* Information and Computation 93, 55–92, (1986).
21. C.C. Morgan. *Programming from Specifications*, Prentice-Hall, Second edition, (1994).
22. C.C. Morgan, A. McIver, K. Seidel and J.W. Sanders. *Refinement-oriented Probability for CSP*, Technical Report PRG-TR-12-94, Oxford University Computing Laboratory, (1994).
23. C.C. Morgan, A. McIver, K. Seidel and J.W. Sanders. *Probabilistic predicate transformers* Technical Report PRG-TR-5-95, Oxford University Computing Laboratory, (1995).
24. J.M. Morris. *A theoretical basis for stepwise refinement and the programming calculus*, Science of Computer Programming, 9(3): 287-306, (1987).
25. J.R. Rao. *Reasoning about probabilistic parallel programs.* ACM Transactions on Programming Language and Systems, 16(3), (1994)
26. M. Sharir, A. Pnueli and S. Hart. *Verification of probabilistic programs.* SIAM Journal on Computing 13(2), 292–314, (1984).