# Probabilistic Choice, Reversibility, Loops, and Miracles

Bill Stoddart[1], Pete Bell[2]

[1] University of Teesside, UK
[2] Consultant

**Abstract.** We consider an addition of probabilistic choice to Abrial's Generalised Substitution Language (GSL) in a form that accommodates the backtracking interpretation of nondeterministic choice. Our formulation is introduced as an extension of the Prospective Values formalism we have developed to describe the results from a backtracking search. Significant features are that probabilistic choice is governed by feasibility, and nontermination is strict. The former property allows us to use probabilistic choice to generate search heuristics. In this paper we are particularly interested in iteration. By demonstrating sub-conjunctivity and monotonicity properties of expectations we give the basis for a fixed point semantics of iterative constructs, and we consider the practical proof treatment of probabilistic loops. We discuss loop invariants, loops with probabilistic behaviour, and probabilistic termination in the context of our formalism, which is strict with respect to nontermination. The formal constructs described are incorporated in a reversal virtual machine (RVM).
**Keywords.** B Method, probabilistic choice, prospective values, pGSL, bunches, backtracking.

## 1 Introduction

Probabilistic algorithms exist for many applications, with some well known examples being primality testing (Rabin's algorithm), Quicksort with random pivot selection (which has optimum expected performance against a hostile oracle), Buffon's algorithm for the evaluation of $\pi$, Quantum algorithms such as Shor's algorithm, and randomised back-off algorithms for resolving symmetric choice [12]. This has motivated researchers to add probabilistic choice to formalisms which underpin formal software development, such as GCL, GSL, and Hoare-He Designs,

Tractable formulations for doing this have not been easy to find. A major difficulty has been in the interaction between non-deterministic and probabilistic forms of choice, and this is seen in all approaches. The semantics of probabilistic programs was first formulated using measure theoretic approaches by Kozen[10]. A more immediately practical approach based on pGCL (the Guarded Command Language extended with probabilistic choice) has been developed by He, Morgan, McIver, Sanders and others[14, 15, 4]. A discursive exposition of this approach is

available in the monograph of McIver and Morgan[11]. Hurd[7, 8] has developed an approach based on a shallow embedding of probabilistic programming concepts in HOL, in which random events are modelled by popping elements from an assumed infinite series of coin flips, and has worked with Morgan and McIver on the mechanisation, in HOL, of probabilistic guarded commands. Meinicke and Hayes [13] have given an extensive account of algebraic properties of probabilistic action systems. The combination of reversibility and probability is addressed by He Jifeng and J Sanders in [9].

In previous work, we have explored the use of nondeterministic choice within search procedures, both in the B Formalism and in terms of Hoare and He's unifying theories [20]. We propose the formalism $S \diamond E$ to represent all the values expression $E$ might take after executing the program $S$. By adding probabilistic choice to our language we obtain an interpretation of the expectation of $S \diamond E$ as the expected values of "observation" $E$ after conducting "experiment" $S$. We exploit reversibility to support backtracking, and in our approach both probabilistic and nondeterministic choice are "governed by feasibility". By this we mean that if a choice subsequently leads to an infeasible continuation, execution will backtrack to the point of choice and try an alternative.

Abrial's Generalised Substitution Language provides a suitable vehicle for representing computations based on UTP Designs minus Healthiness Condition H4, which expresses the Law of the Excluded Miracle. In particular GSL includes naked guarded commands, able to express miraculous behaviour. In the context of reversible computations, an infeasible operation will simply cause execution to engage reverse gear, recommencing forward execution when it encounters an unexplored choice.

Central to our project is the provision of an execution platform for the constructs we investigate in the form of the "Reversible Virtual Machine".[16] We propose a programming language with an extended expression syntax which includes terms of the form $S \diamond E$; this yields the value (or bunch of values) $E$ would take after executing $S$, but *does not change the system state.* Operationally it represents the execution of $S$, the recording of the value of $E$, and the restoration of the previous system state by a stepwise reversal of the computation of $S$. This method of organising a computation, and in particular stepwise reversibility, has a thermodynamic significance: the requirements for power consumption in a computation arise from the damping required to reconcile previously incompatible system states, a requirement that is not present if computations are organised in a stepwise reversible manner. Our reversible execution platform in a virtual machine implemented on non-reversible technology, and therefore offers none of the advantages of reversibility in terms of power consumption. However, reversibility has other advantages which we can exploit, for example in terms of garbage collection [2] and in providing a number of new programming structures [20].

The theory of probabilistic programming given here is more fully described in our paper "A Unification of Probabilistic Choice within a Design-based Model of Reversible Computation"[17] and in an associated technical report[18]. The

original contributions of the current paper are the re-expression of our theory in GSL, the establishment of a semantic foundation for probabilistic iterations in our formalism based on fixed point theory, the practical proof treatment of loops with probabilistic loop bodies, and consideration of probabilistic loop termination within a formalism which has a strict approach to probabilistic termination, and some reflections on the total correctness abstraction in the context of backtracking and probabilistic choice. Aspects of our approach which are covered in our previous report [17] include: the ability to derive a relational model from a probabilistic program text, the expression of blind nondeterminism within the same model as demonic nondeterminism, the characterisation of probabilistic refinement as containment of convex closures in distribution space, and the linking of probabilistic and non-probabilistic models via a Galois connection.

The paper is organised as follows. In Section 2 we review our Prospective Values formalism; in Section 3 we introduce probabilistic choice and review our previous work on expectations; in section 4 we consider some algebraic properties of expectations, establishing semi-conjunctivity and monotonicity; in section 5 we establish the basis for a fixed point semantics of iterative constructs; in section 6 we discuss practical proof treatment of loops and probabilistic termination; in section 7 we draw our conclusions and discuss future work.

## 2 Backtracking and Prospective Values

In [21] we introduce $S \diamond E$, to represent the bunch[5, 6] of values that could be taken by expression $E$ after executing the program $S$. The binding power of $\diamond$ is below that of program connectives ( $[\!]$ , $\longrightarrow$ etc ).

We remind the reader that, in Hehner's Bunch Theory, a bunch is "the contents of a set"; thus $1, 2$ is the bunch of elements which are in the set $\{1, 2\}$, and the comma is now an operator, signifying bunch union. The empty bunch is written as **null**. We write $E : F$ to express bunch containment, for example $1, 2 : 1, 2, 3$. Some simple but important properties are $E : E$, $E, F = F, E$, $E : E, F$, **null** $: E$.

Operators applied to bunches are lifted. For example if $A = 1, 2$ and $B = 4, 5$, then $A + B = 1 + 4, 1 + 5, 2 + 4, 2 + 5$.

We define the guarded bunch $g \longrightarrow S$ to have the value of $S$ where $g$ holds, and to equal the null bunch elsewhere.

In our approach we add an improper bunch $\perp$, (more strictly an improper bunch for each type) to represent the value of an expression after a nonterminating computation. For any other bunch $E$ of the same type we have $E : \perp$ and $\neg \perp : E$. The improper bunch has a number of absorptive properties, e.g. $E, \perp = \perp$, $E + \perp = \perp$, $E * \perp = \perp$.

We define the pre-conditioned bunch $P \mid E$ to have the value $E$ where $P$ holds to be equal to the improper bunch elsewhere.

The bunch comprehension $\S\, x \bullet E$ is the bunch of all values taken by $E$ as $s$ ranges over its type, where a type is a maximal set.

The short description covers the notations used in this paper. Full details of our use of Bunch theory are given in [17][1].

We use a large equals $\mathrel{==}$ with the same meaning but lower precedence than $=$. It is particularly useful because $\diamond$ has a lower precedence than $=$. We use $[P]$ to assert that $P$ is true everywhere.

Returning now to the construct $S \diamond E$, in [21] and [20] we define it in terms of the predicative semantics of $S$ and prove that it has the properties:

| Name | Rule | Side Cond |
|---|---|---|
| Precondition | $P \mid S \diamond E = P \mid (S \diamond E)$ | |
| Skip | $skip \diamond E = E$ | |
| Assignment | $x := F \diamond E = E[F/x]$ | |
| Guard | $g \Longrightarrow S \diamond E = g \longrightarrow S \diamond E$ | |
| Choice | $S \,[]\, T \diamond E = S \diamond E, T \diamond E$ | |
| Choice from set | $x :\in A \diamond E = \S\, a \bullet a \in A \longrightarrow E[a/x]$ | $a \setminus E$ |
| Seq Comp | $S;\ T \diamond E = S \diamond T \diamond E$ | |
| Local Variable | $var\ z.S.\ \mathrm{end} \diamond E = \S\, z \bullet S \diamond E$ | $z \setminus E$ |

These rules provide a semantic description of a sequential programming language in a total correctness framework, and for this purpose are of equivalent expressive power to weakest-precondition calculus or Hoare-He designs. They also describe the values that will be taken by terms of the form $S \diamond E$ that occur in the extended expression language of our programs. Note, however, that the RVM does not support the representation of bunches, and where a term of the form $S \diamond E$ represents a non-elementary bunch and occurs in an expression, it must be packaged within set brackets, i.e. as $\{S \diamond E\}$.

## 3 Probabilistic choice and expectations

In [17, 18] we add probabilistic choice to our programming language. In common with pGSL, pGCL and other formalisms we use $S \,{}_p\!\oplus\, T$ for the operation that will choose $S$ with probability $p$ and $T$ with probability $1-p$. Thus the following program represents an experiment in which a coin is tossed two times and the number of heads is recorded in the variable $X$.

$$
\begin{array}{l}
Experiment \mathrel{\widehat{=}} X := 0; \\
\quad X := X + 1 \,{}_{0.5}\!\oplus\, skip; \\
\quad X := X + 1 \,{}_{0.5}\!\oplus\, skip
\end{array}
$$

The expression $X$ has its value assigned according to a random process; it complies with our intuitive understanding of a "random variable". [2] We reason about

[1] Available from http://tees.openrepository.com

[2] Though it is not a random variable as formulated in classical probability theory, where random variables are real valued functions on a sample space, and its random properties are implied from the probability measure over this space.

expressions that can take random values following some operation in terms of expectations.

We write the expected value of an expression $E$ after performing a computation $S$ as $\mathbf{E}(S \diamond E)$, defined according to the following rules:

| Name | Rule | Side Cond |
|---|---|---|
| Precondition | $\mathbf{E}(P \mid S \diamond E) = P \mid \mathbf{E}(S \diamond E)$ | |
| Skip | $\mathbf{E}(\mathit{skip} \diamond E) = E$ | |
| Assignment | $\mathbf{E}(x := F \diamond E) = E[F/x]$ | |
| Guard | $\mathbf{E}(g \Longrightarrow S \diamond E) = g \longrightarrow \mathbf{E}(S \diamond E)$ | |
| Choice | $\mathbf{E}(S \,[\!]\, T \diamond E) = \mathbf{E}(S \diamond E), \mathbf{E}(T \diamond E)$ | |
| Choice from set | $\mathbf{E}(x :\in A \diamond E) = \S\, a \bullet a \in A \longrightarrow E[a/x]$ | $a \setminus E$ |
| Seq Comp | $\mathbf{E}(S;\ T \diamond E) = \mathbf{E}(S \diamond \mathbf{E}(T \diamond E))$ | |
| Local Variable | $\mathbf{E}(\mathit{var}\ z.S.\mathit{end} \diamond E) = \S\, z \bullet \mathbf{E}(S \diamond E)$ | $z \setminus E$ |
| Prob Choice | $\mathbf{E}(S \ {}_p\!\oplus\ T) \diamond E = \mathbf{E}(S \diamond E) \ {}_p\!+\ \mathbf{E}(T \diamond E)$ | $0 < p < 1$ |

Now that we have generalised our approach to include probability, the use of expectations limits us to considering real valued expressions. We can also accommodate vectors of real valued expressions, but the reader may wonder whether we have reduced our power of expression through limiting our interest to real values. This is not, however, the case, as we can capture the state of variables of any type through numerotized predicates. We define $\mid Q \rangle \mathrel{\widehat{=}} Q \longrightarrow 1, \neg\, Q \longrightarrow 0$, so that the value of $\mid Q \rangle$ will be 1 if $Q$ is true and 0 if $Q$ is false.

Our formulation of expectation is based on a weighted addition. However, since we allow our programs to contain infeasible operations, we have to be able to express the effect of a probabilistic choice made between a feasible operation and an infeasible one. We make the implementation choice, in the case of our RVM, that when execution backtracks out of a probabilistic choice that has proved infeasible, it will take the alternative choice, just as it does in the case of provisional non-deteministic choice. We encapsulate this effect within our weighted addition. The define the weighted bunch addition $E_1 \ {}_p\!+\ E_2$ where $E_1$ and $E_2$ are bunches and $p$ is an element with $0 \leq p \leq 1$ by

$$E_1 \ {}_p\!+\ E_2 \mathrel{\widehat{=}} E_1 = \mathbf{null} \longrightarrow E_2 \ ,\ E_2 = \mathbf{null} \longrightarrow E_1 \ ,\ p * E_1 + (1 - p) * E_2$$

The body of this key definition consists of the bunch union of three terms. The definition covers nine cases, these being that each of $E_1$ and $E_2$ could be a proper non-empty bunch, or **null**, or $\bot$. Where $E_1$ and $E_2$ are non-empty the first two terms equate to **null** and thus do not contribute to the result, which is given by the third term. If either $E_1$ or $E_2$ is **null**, then, by the absorptive properties of **null**, the third term will be **null** and the result will be given by the first two terms, at most one of which will be non-null. If either $E_1$ or $E_2$ is $\bot$, the third term of will be $\bot$ (by the absorptive power of $\bot$), and the whole expression will equate to $\bot$.

As an (unsatisfactory) alternative we might have used the rule: $\mathbf{E}(S \ {}_p\!\oplus\ T) \diamond E = p * \mathbf{E}(S \diamond E) + (1 - p) * \mathbf{E}(T \diamond E)$ which would be correct in the case of feasible $S$ and $T$, but would have the unwanted effect of making our formalism strict with respect to feasibility, i.e. a possibly infeasible operation would *certainly*

be infeasible. That is the case in pGSL, but we must reject it as a possible formulation because its implementation in a programming environment which includes possibly infeasible commands would require all branches to be tested for feasibility. We prefer the view that execution will resolve possible infeasibility by use of a backtracking mechanism, and the rule we adopt makes $magic \,{}_p\!\oplus\, skip = skip$. Thus *magic* is a zero element with respect to our probabilistic choice, just as it is with respect to nondeterministic choice, and we even have $S \,{}_0\!\oplus\, magic = S$; thus probabilistic choice, like demonic choice, is governed by feasibility.

A property of our probabilistic choice is that it is strict with respect to nontermination. We define $trm(S)$ as $\mathbf{E}(S \diamond null) : null$. The idea here is that the only way the expected value of the null bunch after running $S$ can be larger than the null bunch is if we cannot guarantee termination of $S$. As an example of how this works consider $abort \mathrel{\widehat{=}} false \mid skip$ and let $S$ be the program $abort \,{}_{0.5}\!\oplus\, skip$ Then we have:

$$\begin{aligned}
&\mathbf{E}(S \diamond null) = \mathbf{E}(abort \,{}_{0.5}\!\oplus\, skip \diamond null) \\
&= \mathbf{E}(abort \diamond null) \,{}_{0.5}\!+\, \mathbf{E}(skip \diamond null) \\
&= \mathbf{E}(false \mid skip \diamond null) \,{}_{0.5}\!+\, null \\
&= \mathbf{E}(false \mid skip \diamond null) = false \mid null \\
&= \bot
\end{aligned}$$

hence

$trm(S) = \bot : null = false$ Using the lenient approach to termination of pGSL or pGCL, the above program would terminate with probability 0.5.

A possible advantage of a strict treatment of nontermination is that use of an operation outside of its pre-condition would be easier to detect during the discharge of proof obligations. The disadvantage is that vanishingly small probabilities of nontermination will dominate our expectations, and we discuss a case later. He and Sanders[9] have engineered a version of pGCL which is strict with respect to non-ermination. To do this they introduce angelic choice and view an operation $S$ as an angelic choice over a set of "fibres". The fibre at $x_0$ behaves like $S$ for $x = x0$ and like abort elsewhere.[9]. We achieve strictness more simply from the properties of the improper bunch.

If $A$ is a predicate on the state space we use $Prob_S(A)$ for the probability that $A$ is true after executing $S$, defined as $Prob_S(A) = \mathbf{E}(S\diamond \mid A\rangle)$. Where $S$ is nondeterministic the result may be a non-elemental bunch, and we will typically be interested in the minimum probability of obtaining some desired result $A$.

In addition to the expected value of a random expression we can also know something about its possible values, as expressed in the following bunch inclusion:

$(S \,{}_p\!\oplus\, T \diamond E) : S \diamond E, T \diamond E$

Using this rule we can show, for example $(x := 0 \,{}_p\!\oplus\, x := 1 \diamond x) : 0, 1$. This shows why we insist on the difference between $S \diamond E$ and $\mathbf{E}(S \diamond E)$ in our notation, since in this case $\mathbf{E}(x := 0 \,{}_x\!\oplus\, := 1 \diamond x) = 1/2$. This is also a good point to emphasize that $\mathbf{E}(S \diamond E)$ is a *composite notation*, i.e. it is *not* evaluated by first calculating $S \diamond E$ and than applying the expectation operator $\mathbf{E}$. An alternative notation which does not allow this possible misunderstanding to occur

would have been $\mathbf{E}(S, E)$. However, we prefeer a notation that emphasised the connection between $S \diamond E$,representing an actual value of a random expression, and $\mathbf{E}(S \diamond E)$, representing the expected value of the same expression.

In order to make probabilistic choices we require a source of random numbers, which must be provided in a computationally reversible manner. We can do this quite simply, by drawing random numbers from a sufficiently large and pre-initialised random-number table. As each number is drawn, a pointer is incremented, modulo the size of the table, so that the next number will be drawn when a random value is next required. The required pointer incrementation is a reversible computation, but we do not ever reverse it. This ensures that there are no correlations between the random events generated in different branches of a backtracking computation. It also means that where $S$ contains random choice, the bunch $S \,[\!]\, S \diamond E$ may consist of different, and generally more, values than $S \diamond E$.

The expectations defined so far are not enough to extract all available information on post distributions. For example, if we define $S \mathrel{\widehat{=}} x := a \,{}_p\!\oplus\, x := b$ and $T \mathrel{\widehat{=}} x := c \,{}_p\!\oplus\, x := d$, and we consider the expectations asociated with $S \,[\!]\, T$, then we can obtain the post-probability that $x = a$ as equal to $0, p$, and similarly the post-probabilities that $x = b$, $x = c$ and $x = d$ as equal to $0, 1-p$ and $0, p$ and $0, 1-p$ respectively. However, this does not enable us to untangle the two distributions. To resolve this we extend our notion of weighted choice in an obvious way to equal length sequences of real values, enabling us to calculate:

$$\mathbf{E}(S \,[\!]\, T) \diamond \langle\!|\, x = a \,|, |\, x = b \,|\rangle\!\rangle = \langle p, 1-p\rangle, \langle 0, 0\rangle$$

from which we see that there is a non-deterministic choice that gives a post distribution for $x$ in which $x$ has probability $p$ to equal $a$ and $1-p$ to equal $b$. The other term in the expectation, that is $\langle 0, 0\rangle$, shows the presence at least one other distribution which has zero probability of setting $x$ to $a$ or $b$.

## 4 Algebraic properties of our expectation calculus

Predicate transformers in B-GSL are conjunctive. i.e. $wp(S, Q_1 \wedge Q_2) = wp(S, Q_1) \wedge wp(S, Q_2)$. I.e. $S$ will establish post condition $Q_1 \wedge Q_2$ exactly when it will establish $Q_1$ and will also establish $Q_2$.

The equivalent property in PV semantics is $S \diamond E, F \;=\; (S \diamond E), (S \diamond F)$.

This correspondence can be illustrated by the use of the conjunctivity property in establishing that sequential composition distributes through choice, i.e. $S;\; T \,[\!]\, U \;=\; (S;\; T) \,[\!]\, (S;\; U)$. In wp semantics we establish the equality of program expressions $S$ and $T$ by showing $wp(S, Q) = wp(T, Q)$ for arbitrary $Q$, Thus to establish the given distributivity rule we proceed as follows:

$wp(S;\ T \,[\!]\, U, Q) =$"wp rule for sequential composition"
$wp(S, wp(T \,[\!]\, U, Q)) =$"wp rule for choice"
$wp(S, wp(T, Q) \wedge wp(U, Q)) =$"wp conjunctivity"
$wp(S, wp(T, Q)) \wedge wp(S, wp(U, Q)) =$"wp rule for sequential composition"
$wp(S;\ T, Q) \wedge wp(S;\ U, Q) =$"wp rule for choice"
$wp((S;\ T) \,[\!]\, (S;\ U), Q)$

and hence $S;\ T \,[\!]\, U\ =\ (S;\ T) \,[\!]\, (S;\ U)$

using PV semantics we establish the equality of $S$ and $T$ by showing the equivalence of their prospective value effect, i.e. that for an arbitrary expression $E$ defined on the current state, that $(S \diamond E) = (T \diamond E)$. To establish the given distributivity rule in PV semantics we proceed as follows:

$S;\ T \,[\!]\, U \diamond Q\ =$ "pv rule for sequential composition"
$S \diamond (T \,[\!]\, U \diamond Q)\ =$ "pv rule for choice"
$S \diamond (T \diamond Q\ ,\ U \diamond Q)\ =$ "pv conjunctivity"
$(S \diamond T \diamond Q)\ ,\ (S \diamond U \diamond Q)\ =$ "pv rule for sequential composition"
$(S;\ T \diamond Q), (S;\ U \diamond Q)\ =$ "pv rule for choice"
$(S;\ T) \,[\!]\, (S;\ U) \diamond Q$

and again we establish our result. We notice that the appeal to the respective conjunctivity properties is made at the same point in both these proofs.

One important property of conjunctivity is that it implies monotonicity, which is a pre-requisite for establishing a fixed point semantics of loops. We would formulate the conjunctivity property for expectations as:

$$\mathbf{E}(S \diamond (A, B))\ =\ \mathbf{E}(S \diamond A), \mathbf{E}(S \diamond B)$$

Our expectation calculus, however is not conjunctive, or more exactly is only conjunctive for operations that do not include probabilistic choice. We see we do not in general have conjunctivity from the following counter example.

Let $S \mathrel{\widehat{=}} x := 0\ {}_{0.5}{+}\ x := 1$ then applying the rules for probabilistic choice to $\mathbf{E}(S \diamond x, x + 1)$ we have

$$\mathbf{E}(S \diamond (x, x + 1))\ =\ 0.5, 1, 1.5$$

but

$$\mathbf{E}(S \diamond x), \mathbf{E}(S \diamond x + 1) = 0.5, 1.5$$

We can however formulate a sub-conjunctivity property. We follow Hehner in defining a bunch refinement $A \sqsubseteq B \mathrel{\widehat{=}} B : A$.

**Theorem 1. Sub-conjunctivity of expectations**

$$\mathbf{E}(S \diamond (A, B))\ \sqsubseteq\ \mathbf{E}(S \diamond A), \mathbf{E}(S \diamond B)$$

**Proof**

The proof is by structural induction with base cases for *skip* and assignment and proofs for each program connective, these making appeals to the inductive case. Here we give just the base case for assignment and the inductive proof for probabilistic choice.

For assignment we have:

$\mathbf{E}(x := E \diamond (A, B)) \;=\;$ "expectation rule for assignment"
$(A, B)[E/x] \;=\;$ "distributivity of substitution through bunch union"
$A[E/x], B[E/x] \;=\;$ "expectation rule for assignment"
$\mathbf{E}(x := E \diamond A), \mathbf{E}(x := E \diamond B) \;\sqsubseteq$ "property of bunch refinement"
$\mathbf{E}(x := E \diamond A), \mathbf{E}(x := E \diamond B)$

For probabilistic choice with $0 < p < 1$ we have:

$\mathbf{E}(S_p \oplus T \;\diamond\; (A, B)) \;=\;$ "rule for prob choice"
$\mathbf{E}(S \;\diamond\; (A, B)) \;{}_p{+}\; \mathbf{E}(T \;\diamond\; (A, B)) \;\sqsubseteq$ "inductive case and property of ${}_p{+}$"
$(\mathbf{E}(S \;\diamond\; A), \mathbf{E}(S \;\diamond\; B)) \;{}_p{+}\; (\mathbf{E}(T \;\diamond\; A), \mathbf{E}(T \;\diamond\; B)) \;=$
"lifted application of ${}_p{+}$"
$\mathbf{E}(S \;\diamond\; A) \;{}_p{+}\; \mathbf{E}(T \;\diamond\; A), \mathbf{E}(S \;\diamond\; A) \;{}_p{+}\; \mathbf{E}(T \;\diamond\; B),$
$\mathbf{E}(S \;\diamond\; B) \;{}_p{+}\; \mathbf{E}(T \;\diamond\; A), \mathbf{E}(S \;\diamond\; B) \;{}_p{+}\; \mathbf{E}(T \;\diamond\; B) \;\sqsubseteq$
"defn of bunch refinement"
$\mathbf{E}(S \;\diamond\; A) \;{}_p{+}\; \mathbf{E}(T \;\diamond\; A), \mathbf{E}(S \;\diamond\; B) \;{}_p{+}\; \mathbf{E}(T \;\diamond\; B) \;=$
"rule for prob choice"
$\mathbf{E}(S_p \oplus T \;\diamond\; A), \mathbf{E}(S_p \oplus T \;\diamond\; B)$

Other cases follow in an obvious way. □

We now return to the subject of monotonicity, which fortunately is implied by sub-conjunctivity.

**Theorem 2. Monotonicity of expectations**

$$[A \;\sqsubseteq\; B] \Rightarrow \mathbf{E}(S \;\diamond\; A) \;\sqsubseteq\; \mathbf{E}(S \;\diamond\; B)$$

We use the following obvious lemma

**Lemma 1.** *Bunch refinement lemma*

$$[A \;\sqsubseteq\; B] \;\Rightarrow\; A \;=\; A, B$$

**Proof** We must prove $\mathbf{E}(S \diamond A) \;\sqsubseteq\; \mathbf{E}(S \diamond B)$ under the assumption $[A \;\sqsubseteq\; B]$

$\mathbf{E}(S \;\diamond\; A) \;=\;$ "assumption and referential transparency"
$\mathbf{E}(S \;\diamond\; (A, B)) \;\sqsubseteq$ "semi-conjunctivity of expectations"
$\mathbf{E}(S \;\diamond\; A), \mathbf{E}(S \;\diamond\; B) \;\sqsubseteq$ "defn of bunch refinement"
$\mathbf{E}(S \;\diamond\; B)$

□

The monotonicity of expectations will be of use in the next section, when it allows us to infer the monotonicity of a function used in a fixed point equation to characterise the transitive opening of an operation, and which we subsequently use to define the meaning of a while loop in terms of expectation calculus.

## 5 Expectations and iterative commands

In this section we are concerned with asking whether it makes any mathematical sense to talk about an expectation of some expression following an iterative command, and how such commands may be defined in terms of the basic table of commands for which expectation rules have been given in Section 3. We construct an argument based on fixed point theory, following the approach Abrial takes in [1] to justify iterative constructs in a predicate transformer context. We first define the "transitive opening" of an operation, and examine its expectation properties. We then use transitive opening as the basis for defining a while loop.

### 5.1 Expectations and transitive opening

We make use of Abrial's definition of the transitive opening $S^\wedge$ of a command $S$, defined as:

$$S^\wedge \mathrel{\widehat{=}} \mu\, X.(X;\ S) \,[]\, \mathit{skip}$$

In this definition of $S^\wedge$ in terms of a fixed point equation, the associated cpo is the lattice of operations, with top element magic and bottom element abort, and the ordering is reverse refinement. We choose the weakest fixed point to include infinite behaviour. The corresponding strongest fixed point would give us $S^*$, the transitive closure of $S$.

We first prove the following

**Lemma 2.** *The expectation effect of transitive opening*

$$\mathbf{E}(S^\wedge \diamond E) \;=\; \mu\, Y.\mathbf{E}(S \diamond Y), E$$

**Proof** From the definition of $S^\wedge$ we have that $S^\wedge$ is the least (least refined) operation that satisfies

$$S^\wedge \;=\; (S;\ S^\wedge) \,[]\, \mathit{skip}$$

Taking expectations (and by referential transparency)

$\mathbf{E}(S^\wedge \diamond E) \;=\; \mathbf{E}((S;\ S^\wedge) \,[]\, \mathit{skip} \diamond E) \;=$
"expectation rule for choice"
$\mathbf{E}(S;\ S^\wedge \diamond E), \mathbf{E}(\mathit{skip} \diamond E) \;=$
"expectation rules for sequential composition and skip"
$\mathbf{E}(S \diamond \mathbf{E}(S^\wedge \diamond E)), E$

Thus we obtain the following fixed point equation for $\mathbf{E}(S^\wedge \diamond E)$

$$\mathbf{E}(S^\wedge \diamond E) \;=\; \mathbf{E}(S \diamond \mathbf{E}(S^\wedge \diamond E)), E$$

We have thus transformed a fixed point equation on operations to a fixed point equation on expectations. Our cpo is now the lattice of bunches of values

that can be taken by expectations , and our order is reverse bunch refinement, with top element *null* and bottom element the improper bunch $\perp$. Once again, to include infinite behaviour we take the least solution, giving

$$\mathbf{E}(S^{\wedge} \diamond E) = \mu Y.\mathbf{E}(S \diamond Y), E$$

We may assure ourselves that such a fixed point indeed exists by appeal to Tarski's fixed point theorem. This states that an equation of the form $X = f(X)$ will have solutions if the domain of $f$ is a cpo and $f$ is monotonic. in our case the function $f$ is given by $f(Y) = \mathbf{E}(S \diamond Y), Y$ the domain of $f$ is a lattice (and therefore a cpo) and the monotonicity of $f$ is assured by the monotonicity property of expectations, proved in the previous section.
□

### 5.2 WHILE loops

We now take Abrials definition of a while loop and again we will investigate its effect on expectations, showing that it gives rise to a well defined fixed point, and that the usual unwinding interpretation of a while loop still holds. We define:

$$\text{while } G \text{ do } S \text{ end } = (G \Longrightarrow S)^{\wedge};\ G \Longrightarrow \mathit{skip}$$

And the following theorem describes the effect of a while loop within the expectation calculus.

**Theorem 3.** *Fixed point interpretation of post loop expectations.*

$$\mathbf{E}(\text{ while } G \text{ do } S \text{ end } \diamond E) = \mu Y.\ \text{if } G \text{ then } \mathbf{E}(S \diamond Y) \text{ else } E \text{ end}$$

**Proof** We consider the expectation effect of a while loop on an arbitrary expression.
$\mathbf{E}(\text{ while } G \text{ do } S \text{ end } \diamond E)$ = "defn of while loop"
$\mathbf{E}((G \Longrightarrow S)^{\wedge};\ \neg G \Longrightarrow \mathit{skip} \diamond E)$ =
"expectation rule for sequential composition"
$\mathbf{E}((G \Longrightarrow S)^{\wedge} \diamond \mathbf{E}(\neg G \Longrightarrow \mathit{skip} \diamond E))$ =
"expectation rules for guard and skip"
$\mathbf{E}((G \Longrightarrow S)^{\wedge} \diamond \neg G \longrightarrow E)$ = "Lemma 2"
$\mu Y.\mathbf{E}(G \Longrightarrow S \diamond Y), \neg G \longrightarrow E$ = "expectation rule for guard"
$\mu Y.G \longrightarrow \mathbf{E}(S \diamond Y), \neg G \longrightarrow E$ =
"rewriting as a conditional expression"
$\mu Y.$ if $G$ then $\mathbf{E}(S \diamond Y)$ else $E$ end
□

**Corollary 1.** *The unwinding interpretation of a loop expressed in terms of expectations. Writing* while $G$ do $S$ end *as* $W$ *we have*

$$\mathbf{E}(W \diamond E) = \mathbf{E}(\text{ if } G \text{ then } S;\ W \text{ end } \diamond E)$$

**Proof**

$\mathbf{E}(W \diamond E)$ = "from theorem 3"
if $G$ then $\mathbf{E}(S \diamond \mathbf{E}(W \diamond E))$ else $E$ end =
"conditional expression rule"
$G \longrightarrow \mathbf{E}(S;\ W \diamond E), \neg G \longrightarrow \mathit{skip} \diamond E)$ =
"expectation rule for guard"
$\mathbf{E}(G \Longrightarrow S;\ W \diamond E), \mathbf{E}(\neg G \Longrightarrow \mathit{skip} \diamond E)$ =
"expectation rule for choice"
$\mathbf{E}(G \Longrightarrow S;\ W \,[]\, \neg G \Longrightarrow \mathit{skip} \diamond E)$ =
"definition of IF construct"
$\mathbf{E}(\text{ if } G \text{ then } S;\ W \text{ end } \diamond E)$

□

We terminate this section with a note on our choice of the weakest fixed point in our interpretation of loop expectation semantics. This seems intuitively correct, for the same reason that the weakest fixed point is chosen to describe the predicate transformer effect of loop semantics, i.e. to include the infinite case. We now check this intuition for a particular extreme case.

Theorem 3 tells us that $\mathbf{E}(W \diamond E)$ is a solution to the equation

$$\mathbf{E}(W \diamond E) = \text{ if } G \text{ then } \mathbf{E}(S \diamond \mathbf{E}(S \diamond \mathbf{E}(W \diamond E)) \text{ else } E \text{ end}$$

If we set $G$ to true, and thus make a nonterminating loop, and (for simplicity) set $S$ to *skip*, the equation reduces to

$$\mathbf{E}(W \diamond E = \mathbf{E}(\mathit{skip} \diamond \mathbf{E}(W \diamond E))$$

which by the rule for skip reduces to

$$\mathbf{E}(W \diamond E = \mathbf{E}(W \diamond E)$$

an equation which conveys no information and thus admits any solution. However, since we are taking the weakest fixed point as our solution we obtain the improper bunch as the expected value of an expression following the termination of this nonterminating loop, and this is what we expect.

## 6 Practical proof treatment of loops and termination

The preceding section demonstrates that our expectation calculus gives an interpretation of loops which is able to give a mathematical interpretation to the meaning of the expectation of some expression after executing a loop. As with other formalisms, however, practical proof treatment of loops is not based directly on such a treatment, but rather uses a technique in which loop behaviour is characterised in terms of loop variants and invariants, which capture the programmers intuition about what the loop is intended to achieve and why it is sure to terminate.

Treatment of loops and heuristics for finding probabilistic loop invariants follow the approach described by McIver and Morgan in [11] with the exception

that, due to our strict interpretation of nondeterminism, possibly nonterminating loops become definitively nonterminating. We consider first an example in which termination is deterministic but the result achieved is probabilistic and illustrate the loop invariant method for this case. We then consider two contrasting loops with different forms of probabilistic termination.

For our first example we have a sequence of Bernoulli trails and we are interested in the probability distribution of number of successes obtained, i.e. the classical binomial distribution. The following will be referred to as *prog* in subsequent discussion.

$r := 0;\ i := n;\ \ /*\ init\ */$
while $i \neq 0$ do
$\quad r := r + 1\ {}_p\!\oplus\ skip;\ i := i - 1\ /*\ body\ */$
$\quad$ variant $i$
$\quad$ invariant $c(i, k-r) * p^{k-r} * (1-p)^{i-k+r} * \mid 0 \leq k - r \wedge k - r \leq i \mid$
end

We want to show that $prog \diamond r$ follows a binomial $b(n, p)$ distribution, i.e. that for any $k$ with $k \in 0..n$ we have $Prob_{prog}(r = k) = c(n, k) * p^k * (1-p)^{n-k}$. Here $c$ is the binomial coefficient function defined by $c(n, k) = n!/((n-k)! * k!)$.

The probabilistic loop invariant, found by one of the heuristics proposed in [11], has the form $p* \mid pred \mid$. We consider the computation from some general point at which $r$ successes have been achieved and $i$ trials remain. At that point *pred* is the necessary condition it is still possible to finish with $k$ successes, and $p$ is the probability that this will occur assuming *pred*. If the loop invariant can be preserved the value of $Prob_{prog}(r = k)$ is given by $\mathbf{E}(init \diamond I)$ where $I$ is the loop invariant. i.e.

$\mathbf{E}(r := 0;\ i := n\ \diamond\ c(i, k-r) * p^{k-r} * (1-p)^{i-k+r} * \mid 0 \leq k - r \wedge k - r \leq i \mid$
$\quad = \quad c(n, k) * p^k * (1-p)^{i-k} * \mid 0 \leq k \wedge k \leq i \mid$

The loop preservation rule in our style of presentation is:

$\mid g \mid * I \ \Rrightarrow\ \mathbf{E}(body \diamond I)$

where we use $\Rrightarrow$ for "everywhere less than or equal to" (the equivalent to "everywhere implies" when working with numerotized predicates). We also use $\Lleq$ with an obvious similar meaning.

To show the invariant property we reason from the left hand side:

$\mathbf{E}(body \diamond I) =$
$\mathbf{E}(r := r + 1\ {}_p\!\oplus\ skip;\ i := i - 1\ \diamond\ c(i, k-r) * p^{k-r} * (1-p)^{i-k+r} * \mid 0 \leq k - r \wedge k - r \leq i \mid)$
$\quad =$ "seq comp and assignment"
$\mathbf{E}(r := r + 1\ {}_p\!\oplus\ skip\ \diamond\ c(i-1, k-r) * p^{k-r} * (1-p)^{i-1-k+r} * \mid 0 \leq k - r \wedge k - r \leq i - 1 \mid)$
$\quad =$ "prob choice and assignment"
$p * c(i-1, k-r-1) * p^{k-r-1} * (1-p)^{i-k+r} \mid 0 \leq k - r - 1 \wedge k - r \leq i \rangle$
$\quad \mathbf{E}(x := E \diamond A), \mathbf{E}(x := E \diamond B) +$
$(1-p) * c(i-1, k-r) * p^{k-r} * (1-p)^{i-1-k+r} * \mid 0 \leq k - r \wedge k - r \leq i - 1 \mid)$
$\quad =$ "collecting terms"

$c(i-1,k-r-1)*p^{k-r}*(1-p)^{i-k+r} \mid 0 \leq k-r-1 \wedge k-r \leq i \mid$
$\quad +$
$c(i-1,k-r)*p^{k-r}*(1-p)^{i-k+r}* \mid 0 \leq k-r \wedge k-r \leq i-1\rangle)$
$\Leftarrow$ "comparing numerotized predicates"
$(c(i-1,k-r-1)+c(i-1,k-r))*p^{k-r}*(1-p)^{i-k+r}* \mid 0 \leq k-r \wedge k-r \leq i\rangle)$
$=$ "Pascal's Triangle property of binomial coefficients"
$c(i,k-r)*p^{k-r}*(1-p)^{i-k+r}* \mid 0 \leq k-r \wedge k-r \leq i\rangle)$
$\Leftarrow$ " since $\mid i \neq 0\rangle \leq 1$
$\mid i \neq 0\rangle * c(i,k-r)*p^{k-r}*(1-p)^{i-k+r}* \mid 0 \leq k-r \wedge k-r \leq i\rangle)$
$=$ $\mid g\rangle * I$ $\square$

In the next two examples we consider loops which illustrate problems both with our strict approach to nontermination and with the concept of "termination with probability one". First we consider a loop for which termination is always possible and where the probability of termination in the first or first few iterations may be arbitrarily close to one. Nevertheless we will be able to show that there is a non-zero probability of nontermination of this loop. On termination the loop will leave a variable $i$ set to the number of iterations performed. We will be able to derive the probability that $i = k$, but we will not be able to derive, via our formalism, a numeric value for the expected value of $i$. In the second example we will have a very different situation: a loop that can easily be shown to terminate with probability one but for which, in practice, it would be very imprudent to assume that termination would occur in any human time scale.

The first makes use of the binomial trials described above, which we now assume are packaged in an operation *Bin*, which inputs a number of trials to be performed and the probability of success on each trial, and outputs the number of successful trials.

$r \leftarrow Pterm1(p) \mathrel{\widehat{=}} 0 < p \wedge p < 1 \mid$
  $k := 1;\ i := 0;$
  while $k \neq i$ do
    $i := i + 1;\ k \leftarrow Bin(i,p)$
  end
  $r := i;$

At the *ith* iteration the program performs $i$ Bernoulli trials and terminates if all are successful. The probability of termination on the 1st iteration is $p$. Termination on the second iteration occurs only if we have nontermination on the first iteration followed by termination on the second. It has probability $(1-p)*p^2$. Probability of termination on the third iteration is $(1-p)*(1-p^2)*p^3$ and so on. It seems we should be able to show, within our formalism, that the expected number of iterations required for termination is:
$\mathbf{E}(r \leftarrow Pterm1(p)\ \diamond\ r) = p + 2*(1-p)*p^2 + 3*(1-p)*(1-p^2)p^3 + ..$
The reason we cannot do so is that termination is not guaranteed. The probability of nontermination is given by the infinite product whose *ith* term is the probability that termination does not occur at the *ith* iteration, i.e. $P(p) = (1-p)*(1-p^2)*(1-p^3)....$ For $p = 1/2$ the value of this product has a known analytic form and its value is given in [3] as approximately 0.288788. As we in-

crease $p$ from $1/2$ and approach 1 we can make the probability of termination on the first iteration as close to 1 as we like. We therefor wonder if there is some value $p_0$ for $p$ with $p_0 < 1$ but $P(p_0) = 0$. In fact this cannot be the case, and we will always have a finite probability of nontermination for our loop. As a first step in showing why this must be so assume some $p_0$ exists with $P(p_0) = 1$ and let it also be the smallest such value. Then consider $p_0^2$. We have $p_0^2 < p_0$, so if we can show $P(p_0^2) = 0$ we will have contradicted our assumption that $p_0$ is the smallest value with this property and thus have shown that no such smallest value exists. We have:
$P(p_0^2) = (1 - p_0^2) * (1 - p_0^4) * (1 - p_0^6)...$ = "since $(1 - x^2) = (1 - x) * (1 + x)$"
$(1-p_0)*(1-p_0^2)*(1-p_0^3)...(1+p_0)*(1+p_0^2)*(1+p_0^3)...$ = "since $P(p_0) = 0$"
$0 * (1 + p_0) * (1 + p_0^2) * (1 + p_0^3)...$
This will be zero so long as $(1 + p_0) * (1 + p_0^2) * (1 + p_0^3)...$ has a finite value, and this will be the case so long as $log((1 + p_0)(1 + p_0^2)(1 + p_0^3)...)$ has a finite value. Recalling that $log(1 + x) = x - x^2/2 + x^3/3..$ and noting that therefore $0 < x < 1 \Rightarrow log(1 + x) < x$ we have
$log((1 + p_0) * (1 + p_0^2) * (1 + p_0^3)..) = log(1 + p_0) + log(1 + p_0^2) + log(1 + p_0^3)... <$
$p_0 + p_0^2 + p_0^3 + ...$ = "standard geometric progression"
$p_0/(1 - p0)$ which has a finite value for $p_0 \in openinterval(0, 1)$
Thus we conclude there is no smallest $p$ such that $P(p) > 0$. That is not yet enough to prove that we cannot have $p \in openinterval(0, 1) \wedge P(p) = 0$, but since $P(p)$ is continuous and monotonic decreasing, the only remaining possibility is that the region for which $P(p) > 0$ runs up to and includes some $p_1 \in openinterval(0, 1)$. Then we would have $P(p_1) > 0$ but $p \in openinterval(p_1, 1) \Rightarrow P(p) = 0$. However, we can also dispose of this possibility: if such a $p_1$ exists let $p_2 = p_1 + \epsilon$, and we can show in an obvious way that we can choose $\epsilon$ such that $p_2^2 < p_1$.

Having concluded that the operation $Pterm1$ will always have some finite probability of nontermination, given by $P(p)$, we return to our expression of the expected value for the number of loop iterations in $Pterm1$ but we now include this term and calculate its effect:
$\mathbf{E}(r \leftarrow (Pterm1(p) \diamond r) = p+2*(1-p)*p^2+3*(1-p)*(1-p^2)p^3+..+P(p)*\perp$
= "since $e * \perp = \perp$ for any term $e$
$\mathbf{E}(r \leftarrow (Pterm1(p) \diamond r) = p + 2 * (1 - p) * p^2 + 3 * (1 - p) * (1 - p^2)p^3 + .. + \perp$
= "since $e + \perp = \perp$ for any term $e$" $\perp$
Not surprisingly the absorptive effect of the improper bunch dominates the calculation, however small the term multiplying it might be.

For the second of these two examples we take a loop whose termination is readily provable by the zero one law, but, with suitable chosen parameters, is effectively nonterminating on any practical time scale.
$r \leftarrow Pterm1(p, b) \mathrel{\widehat{=}} 0.5 < p \wedge p < 1 \wedge b > 0 \mid$
$i := 1 \,{}_p\!\oplus\; i := 1;$
while $i \neq 0$ do
$i := i + 1 \,{}_p\!\oplus\; i := i - 1;$
if $i > b$ then $i := b$ end

if $i < -b$ then $i := -b$ end
end
$r := i$;

The program represents a random walk, biased to move in a upward directions and with barriers at $-b$ and $b$. To apply the zero-one law to show termination with probability 1 we need to show the loop always has some probability of termination bounded away from zero. Here, such a bound is given by $(1-p)^b$, which is the probability of terminating in $b$ steps when we are at the upper barrier. Every other position gives a better probability of terminating within the next $b$ steps, and obviously all such probabilities are less that the overall probability of termination from any current position.

We have a trivial application of the zero one law for proving the termination of this loop with probability one and the impossibility of proving termination occurs with probability one in the previous case, but we also have the following: given any integer $N > 0$, however large and some $\epsilon \in \mathit{openinterval}(0, 1)$, however small, we can choose $p$ and $b$ such that the probability of the first loop terminating within $N$ iterations is greater than $1 - \epsilon$ (i.e arbitrarily close to 1) and the probability of the second terminating within $N$ iterations is less that $\epsilon$ (i.e. arbitrarily close to 0). We conclude from this that we need to supplement a proof of termination with probability 1 with a proof that termination is highly likely to occur in some sensible number of iterations given by the contact of the application. Also we need to be aware that loops which look extremely likely to terminate may not, in fact, terminate with probability one, and may need to have termination imposed after some suitable number of iterations, since the slightest possibility of nontermination of a loop will dominate our calculations.

## 7 Conclusions and Future Work

We have considered the fixed point semantics and proof treatment of iterative constructs within an expectation calculus designed to describe reversible computations. To arrive at this calculus from the classical B semantics of sequential programming we remove the law of the excluded miracle, and in terms of UTP Designs, this is equivalent to removing healthiness condition H4. We consider perspective value terms of the form $S \diamond E$ which represent the value expression $E$ would take if S were to be executed. This has a semantic role, but is also a term in our extended language of expressions, which is implemented by executing $S$ , evaluating $E$ , and reversing execution of $S$ to uncompute its effects. Where $S$ is nondeterminstic, this is captured by $S \diamond E$ yielding a bunch of possible values and the corresponding execution of $S \diamond E$ executing all possible branches of the corresponding search tree.

A prospective value calculus allows us to make a smooth transition to an expectation calculus, but we must treat probabilistic choice in a manner that suits the execution behaviour of our reversible machine. This requires magic to be a unit of probabilistic choice, as it is of demonic choice.

Our previous work on expectation gives a relation model and links our prospective value calculus and our expectation calculus by means of a Galois Connection.

In the current paper we interpret Abrials definitions for recursive constructs in terms of our expectation calculus. To provide a fixed point treatment of post loop expectations we had to establish a monotonicity property for our calculus, which we derived by first showing that expectations are sub-conjunctive.

We also consider the practical proof treatment of probabilistic iterations, using the technique of loop variants and invariants, and we show how to look for an appropriate loop invariant. We consider at some length the problem of probabilistic termination, giving examples to show that even loop that seems intuitively certain to terminate may retain a residual possibility of nontermination. Since our approach to termination is strict, we must be on the look our for such possibilities, as they will swamp any terminating behaviour. One the other hend, we also note that termination with probability one is not a strong property, and should be supplemented by knowing how likely termination is after a sensible number of iterations. We finally consider plans for future work. In our present approach, nondeterministic choice plays two roles. It can represent provisional choice, subject to revision by backtracking, or implementor's choice, which may be removed during the refinement process. For some problems it may be more appropriate to replace nondeterministic choice used as provisional choice by a preferential choice, and make this distinct from implementor's choice. We have described a calculus which does this is given in [19]; this also describes in a more concrete way the execution behaviour of certain structures implemented on the RVM. At present this calculus does not incorporate probabilistic choice, and part of our future work will be to perform this additional unification.

## References


1. J-R Abrial. *The B Book*. Cambridge University Press, 1996.
2. H G Baker. The Thermodynamics of Garbage Collection. In Y Bekkers and Cohen J, editors, *Memory Management: Proc IWMM'92*, number 637 in Lecture Notes in Computer Science, 1992.
3. S. R. Finch. *Mathematical Constants*. Cambridge, 2003.
4. Jifeng He, Karen Seidel, and Annabelle McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2-3):171–192, 1997.
5. E C R Hehner. Bunch theory: A simple set theory for computer science. *Information Processing Letters*, 12.1 pp26-31, 1981.
6. E C R Hehner. *A Practical Theory of Programming*. Springer Verlag, 1993. Latest version available on-line.
7. Joe Hurd. *Formal Verification of Probabilistic Algorithms*. PhD thesis, Computer Laboratory, University of Cambridge, 2001.
8. Joe Hurd. A Formal Approach to Probabilistic Termination. In *TPHOLs 2002*, pages 230–245, 2002.
9. He Jifeng and Sanders J. W. Unifying probability. In S. E. Dunne and W Stoddart, editors, *UTP2006 The First International Symposium on Unifying Theories of Programming*, number 4010 in Lecture Notes in Computer Science, 2006.

10. D Kozen. Semantics of Probabilistic Programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.
11. Annabel McIver and Carroll Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems*. Springer, 2004.
12. Annabelle McIver, Carroll Morgan, and Thai Son Hoang. Probabilistic Termination in B. In D Bert, J Bowen, S King, and M Walden, editors, *ZB2003*, number 2651 in Lecture Notes in Computer Science, 2003.
13. L Meinicke and I Hayes. Algebraic reasoning for probabilistic action systems and while-loops. *Acta Informatica*, 45(5), 2008.
14. C. Morgan, A. McIver, K. Seidel, and Sanders J. W. Tr-4-95, probabilistic predicate transformers. Technical report, Oxford University Programming Research Group, 1995.
15. Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996.
16. W. J. Stoddart. The Reversible Virtual Machine. User and technical manuals, 111 pages, University of Teesside, UK, July 2006.
17. W J Stoddart and F Zeyda. A Unification of Probabilistic Choice within a Design-based Model of Reversible Computation. *Formal Aspect of Computing*, 2007. Published on line, DOI 10.1007/s00165-007-0048-1.
18. W J Stoddart and F Zeyda. Probabilistic Choice. Technical report, University of Teesside, UK, 2008. 35 Pages.
19. W J Stoddart, F Zeyda, and S Dunne. Preference and non-deterministic choice. In A Cavalcanti, D Déharbe, M-C Gaudel, and M Woodcock, editors, *ICTAC*, volume 6255 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 2010.
20. W J Stoddart, F Zeyda, and A R Lynas. A Design-based model of reversible computation. In *UTP'06, First International Symposium on Unifying Theories of Programming*, volume 4010 of *Lecture Notes in Computer Science*, June 2006.
21. F Zeyda, W. J. Stoddart, and S Dunne. A Prospective-value Semantics for the GSL. In M Henson, S King, S Schneider, and H Treharne, editors, *ZB2005*, number 3455 in Lecture Notes in Computer Science, 2005.