A Unification of Probablilistic Choice within a Design-based Model of Reversible Computation

University of Teesside Formal Methods and Programming Research Group

Bill Stoddart, University of Teesside
and
Frank Zeyda, University of York

4 March 2008

Abstract

We see reversible computing as a generalisation of sequential computation obtained by revoking the law of the excluded miracle. Our execution language includes naked guarded commands and non-deterministic choice. Choices which lead to miraculous continuations invoke reverse computation, and non-deterministic choice plays the rôle of provisional choice within a backtracking context. We require probabilistic choice for symmetry breaking and sampling large search spaces, but must formulate it differently from previous approaches to obtain the required interactions between probabilistic choice and non-deterministic choice and between probabilistic choice and feasibility. Our formulation allows us to derive the post-distributions which characterise a program, and we use these to construct a relational model. We consider refinement as containment of convex closures within distribution space, qualified with additional conditions to avoid over-refinement. We link the non-probabilistic and probabilistic versions of the model with a Galois connection and show that classical designs are a retract of our probabilistic designs. We consider the interaction between probabilistic and non-deterministic choice and find the same initially counter-intuitive results that have been noted by other investigators. We provide an alternative formulation, within the same model, of oblivious non-determinism, which allows all non-deterministic choices to be moved to the start of a computation. We consider the interaction between probabilistic choice and feasibility that is required to match an operational interpretation in which infeasible commands provoke reverse execution, and we present a small case study to show how the interaction between probabilistic choice and feasibility can be exploited in a practical program. All programming structures described here are supported by our implementation platform, the Reversible Virtual Machine, whose development has accompanied our theoretical investigations.

This technical report is a working document. A peer reviewed version of this work has been accepted for publication in the Formal Aspects of Computing Special Issue on Unifying Theories of Programming.

keywords Reversible Computing; Backtracking; Probability; Hoare-He Designs; Bunches

1 Introduction

In our UTP Symposium paper [SZL06] we investigated, within the UTP framework of Hoare-He designs, the effect of seeing computation as an essentially *reversible* process. We described the theoretical link between reversibility, the physics of computation and minimum power requirements, and we reviewed Paolo Zuliani's work [Zul01] on reversible probabilistic guarded command language. We proposed an alternative formalisation of reversible computing which accommodates backtracking. To obtain a single result from a search we exploited the already recognised properties of non-deterministic choice, using it as provisional choice rather than implementor's choice. We added a prospective-value formalism which can describe programs that return all the possible results of a search, and we showed how to formally describe the premature termination of such a search, a mechanism analogous to the "cut" of Prolog.

In this paper we add probabilistic choice, which we require for symmetry breaking, sampling, and modelling quantum algorithms. Symmetry breaking allows us to resolve ties in search heuristics: multiple runs of the same search algorithm then probabilistically take different execution paths. Replacing provisional choice by probabilistic choice in a large search space allows us to select a random sample of solutions, avoiding the clustering often associated with a set of solutions obtained by imposing a cut. Many additional applications of randomness are described in [MM04]. For a recent sketch of unification of probabilism, reversibility, and quantum computing in a formal context see [HS06]. This work extends the approach of [Zul01] to accommodate, amongst other things, angelic choice, for which the authors find a novel use, namely to provide the option of a strict approach to non-termination, relating their resulting model to **pGCL** by means of a Galois connection. They also consider the implication of providing a language which consists purely of reversible commands in the sense of commands whose corresponding relations are bijections. In this approach there is no additional hidden state. Such a language could be useful to us when providing a concrete description of the instruction set of a reversible virtual machine, where its use would preclude any accidental "cheating" through the introduction of non-reversible commands. When using it, however, one is forced to work at a completely concrete level since the language is not closed under non-deterministic choice.

We turn now to our own approach. As in **pGCL** [MM04] we use $D_1 \ {}_p\!\oplus\ D_2$ to represent a probabilistic choice in which D_1 is chosen with probability p and D_2 with probability $1 - p$. Our formalism differs from that of **pGCL** (which, within a UTP context, we may perhaps take as the standard approach) in a number of ways.

- We take a strict view of non-termination, so that, for example, **abort** $\ {}_{\frac{1}{2}}\!\oplus\ \mathit{II}$ is equivalent to **abort**, whereas in **pGCL** it terminates with probability 1/2. The motivation is to ensure that attempting to discharge proof obligations will clearly signal any precondition violations, even when these occur with only a small probability.

- We take a non-strict view of infeasibility, so that, for example $\mathbf{magic} \underset{\frac{1}{2}}{\oplus} \mathit{I\!I}$ is equivalent to $\mathit{I\!I}$. Whereas the syntactic restrictions of **pGCL** preclude the direct use of naked guarded commands, i.e. commands of the form $g \Longrightarrow D$, they are a vital component of our language, where they are used in conjunction with non-deterministic choice to control backtracking. We describe the interaction between non-deterministic choice, probabilistic choice and feasibility in a manner which captures the execution behaviour of our implementation platform, the Reversible Virtual Machine (RVM), a reversible version of the Forth virtual machine.

- Like **pGCL** we use the statistical idea of expectation to express the properties of random states. However, we take a very general view of what random quantities may be the subject of expectations. We allow any expression of type $\mathbb{R}$, and also any of type $\mathbf{seq}\ \mathbb{R}$, treating them as vectors. Our choice of sequences rather than tuples relates to our interest in infinite sample spaces. By way of contrast, **pGCL** uses expectations based on numerotized predicates.

- In considering the effect of non-deterministic choice, **pGCL** only retains information about the choice least likely to provide a particular postcondition. It is concerned with establishing the minimum guaranteed probability of obtaining that postcondition. In our formalism, where non-determinism plays the dual rôle of expressing both implementor's choice and the provisional choices associated with backtracking, and where we have executable program structures which provide all the possible results of a computation (over its possible non-deterministic choices) we need to retain all the information about such choices. One effect of this is that we can derive post-distributions characterising a program's behaviour.

- There is an interaction, within both formalisms, between probabilistic and non-deterministic assignments to independent variables. This interaction does not reflect the executional reality of our virtual machine. To capture this reality we provide a refined description in which non-deterministic choice is blind with respect to the current state.

For our purposes we need a new variant of the interactions between probabilistic and non-deterministic choice. These interactions are not simple to formulate. He and Sanders comment in [HS06] "The laws relating probabilism and non-determinism are, as we have seen, the most subtle". In a discussion on a programming logic of distributions [MM04], McIver and Morgan find an unexpected result and comment "Because there are several phenomena involved here - and all our pre-conceptions as well - we cannot point to any one of them and say "that causes the contradiction". In a paper which extends a categorical formulation of non-determinism [VWar] Danielle Varacca and Glynne Winskel comment "In Category Theory, non-determinism and probability are represented by suitable monads. These two monads do not combine well, as they are". In his thesis "Probabilistic Extensions of Semantic Models" [dH01] Jerry den Hartog devotes over 100 pages to combining the two forms of choice.

The paper is organised as follows. In Section 2 we discuss mathematical preliminaries. We introduce our adaptation of Eric Hehner's bunch theory, the use of which will greatly simplify the presentation of our theory and in particular

the representation of the combined effects of probabilistic and non-deterministic choice. In this section we also give our precedence rules and parsing conventions. In Section 3 we review our prospective-value formalism, reversibility, and the use of backtracking in non-probabilistic programs. In Section 4 we introduce probabilistic designs, which are designs enhanced by a probabilistic choice operator. We formulate an expectation calculus to study their effect, we give the associated probabilistic relational model, and we show how our expectation calculus can be used to derive the after-state distributions of a probabilistic design. In Section 5 we give a geometric characterisation of our relational model and define refinement in terms of convex closures in distribution space. In Section 6 we relate designs to probabilistic designs via a Galois connection. We show that the former are a retract of the latter and consider the consequences for applying standard (non-probabilistic) reasoning as an abstraction of our probabilistic expectation calculus. In Section 7 we consider the interaction between probabilistic and non-deterministic choice applied to independent sets of variables, and formulate an alternative definition of probabilistic choice that eliminates this interaction so that non-deterministic choice becomes "oblivious". In Section 8 we consider the interaction between probabilistic choice and feasibility. This interaction is of vital importance to the reversible computations aspect of our approach, in which infeasible continuations provoke reverse execution back to the most recent point at which an unexplored choice is available. We include a point search algorithm as a case study which exploits the interaction between probabilistic choice and feasibility. In Section 9 we draw our conclusions and discuss future work.

2 Mathematical Preliminaries

2.1 Bunches

A bunch [Heh81, Heh93] is the "contents of a set" without the packaging that allows set representation to build up nested structures. A bunch of bunches is self-flattening.

Any value is an elementary bunch or element; for example 2 is a bunch. In set theory we make a distinction between 2 and $\{2\}$, i.e. between an element and a set containing just that element. In bunch theory there is no distinction.

The empty bunch is written as **null**. If A and B are bunches then their union, written A, B, is also a bunch. For any bunch A we have $A, \mathbf{null} = A$. We write $A : B$ to say A is a sub-bunch of B. As with sets, the repetition and order of elements has no significance, and thus bunch union is commutative.

Some examples of true predicates that use bunch inclusion are

$2, 3 : 1, 2, 3, 4 \qquad 2 : 2 \qquad A : A, B \qquad \mathbf{null} : 1, 2 \qquad 1, 2, \mathbf{null} : 1, 2$

If A is the bunch $1, 2$ and B is the bunch $3, 5$ then $A + B$ is the bunch made from summing individual values from A and B. Noting that arithmetic operators have a higher precedence than bunch comma, we have

$$A + B = 1 + 3, 2 + 3, 1 + 5, 2 + 5 = 4, 5, 6, 7$$

Standard arithmetic operators applied to bunches of values are all lifted in this way: they distribute through bunch union and are strict with respect to **null**, e.g. $A + \mathbf{null} = \mathbf{null}$.

We sometimes need to write a bunch within brackets to control operator precedence, for example, since + has a higher precedence than comma, we would write $(1, 2) + 3$ for the operation of adding $1, 2$ and 3. This creates a potential conflict with the traditional use of brackets to indicate tuples. In this paper we use the maplet symbol $\mapsto$ to create ordered pairs: brackets are reserved for structuring purposes. Operator precedences will be summarised in Section 2.2.

We write the bunch subtraction of B from A as $A \setminus B$. It represents the elements of A that are not in B.

We adopt bunch theory to our particular ends, which are to use it in a typed (or multi-sorted) theory which uses partial functions together with classical two-valued logic and takes a total-correctness view of program description, i.e. the approach of B and Z, as well as of Hoare-He designs. All variables in our theory denote elements. Bunches only arise as expressions. Bunches have no effect on our treatment of types, which are maximal sets. The type of any non-empty bunch is the same as the type of its elements. We also have an empty bunch of each type. Although all expressions in our formalism are typed, we do not necessarily give the type of each identifier explicitly, requiring only that it can be inferred without ambiguity.

To model non-termination we introduce an improper bunch $\bot$, or more exactly an improper bunch for each type. Given a type (maximal set) T the associated improper bunch is $\bot_T$. Where context can determine its type we just write it as $\bot$. The bunches of any type form a complete lattice under reverse bunch inclusion with **null** and $\bot$ as its top and bottom elements. The properties of the improper bunch are chosen to facilitate the description of sequential computations within a total-correctness framework, i.e within an approach where a computation invoked outside its assumption might provide any result (of the correct type) or fail to terminate. For any proper bunch E we have $E : \bot$ and $\neg\ \bot : E$. Bunch union is strict with respect to $\bot$ i.e. $E, \bot = \bot$. So also is any operation of type $T \nrightarrow T$ or $T \times T \nrightarrow T$. e.g. $E + \bot = \bot$, even when E is **null**. However, it is not strict with respect to maplet construction, so we can have values such as $3 \mapsto \bot$ which are distinguished from $\bot$.

The "guarded bunch" $g \longrightarrow E$ is defined by the property:

$$(g \Rightarrow (g \longrightarrow E = E)) \wedge (\neg\ g \Rightarrow (g \longrightarrow E = \mathbf{null}))$$

and we should note here that we are assuming the use of classical logic in which $g \vee \neg\ g$ is a theorem, so that this property is sufficient to fully define the meaning of a guarded bunch.

The conditional expression **if** g **then** E_1 **else** E_2 **end** is defined by

$$\mathbf{if}\ g\ \mathbf{then}\ E_1\ \mathbf{else}\ E_2\ \mathbf{end}\ \widehat{=}\ g \longrightarrow E_1,\ \neg\ g \longrightarrow E_2$$

The "preconditioned bunch" $p \mid E$ is defined as

$$p \mid E\ \widehat{=}\ \mathbf{if}\ p\ \mathbf{then}\ E\ \mathbf{else}\ \bot\ \mathbf{end}$$

The bunch comprehension $\S\ x \bullet E$, where E is an expression that must include information that determines the type of x, is the bunch of all values that can be taken by E as x ranges over the values of its type. For example $\S\ n \bullet 2 * n$ is the bunch of even numbers, and $\S\ x \bullet 0 < x \wedge x < 3 \longrightarrow 10 * x$ is the bunch $10, 20$.

Given a predicate P we define two further forms of bunch comprehension,

$$\S\, x \mid P \;\widehat{=}\; \S\, x \bullet P \longrightarrow x$$

is the bunch of values of x which satisfy P, and

$$\S\, x \mid P \bullet E \;\widehat{=}\; \S\, x \bullet P \longrightarrow E$$

is the bunch of values taken by E as x ranges over values that satisfy P.

Where a bunch comprehension occurs within set brackets, we omit the bunch comprehension symbol, writing, for example $\{\S\, x \mid P \bullet E\}$ as $\{x \mid P \bullet E\}$, which has its familiar meaning as a set comprehension.

We write $E[F/w]$, where E and F are expressions and w a variable to denote the substitution of F for w in E. If F and w are lists they must be of the same arity and the substitution is made term-wise.

To remain within two-valued logic we avoid bunches of predicates by interpreting inner predicates (membership and equality) in a way that always makes them either true or false. Given expressions X and S of types T and $\mathbb{P}\ T$, the membership predicate $X \in S$ is true if it is point-wise true for each element $x : X$ and $s : S$. Predicates such as $a < b$ are interpreted as set membership, i.e. in this case as $a \mapsto b \in \_ < \_$. Thus $1, 2 < 3$ is true, and both $1, 3 < 3$ and $4 < 3, 5$ are false. Expressions A and B are equal if $A : B$ and $B : A$.

Bunches allow us to define function application in a generalised way. Given $r \in A \leftrightarrow B$ and $a \in A$, and where r, a, A and B are all elementary, we define the application of r to a by:

$$r(a) \;\widehat{=}\; \S\, b \mid a \mapsto b \in r$$

This generalisation of function application renders the separate notion of relational image superfluous, but more importantly it allows us to write $r(x) = y$ (where x and y are elements) to express that r is functional at x and the unique value associated with x in r is y, a luxury not usually permitted in systems which use classical two-valued logic with equality together with a relational model of function application. For example given a partial function f and $f(x) = 3$ we are not entitled, under the classical dispensation, to deduce $x \in \mathbf{dom}\ (f)$ [SDG99, AM02]. With the definition of application given here we can make this deduction, for were it false we would have $f(x) = \mathbf{null}$.

We define the weighted addition $E_1\ {}_p{+}\ E_2$ where p is an element with $0 \leq p \leq 1$ by

Definition 1

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

This is a key definition which will be used in our characterisation of expected values resulting from probabilistic choice. The body of the 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 $\perp$. 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 will be $\bot$ (by the absorptive power of $\bot$), and the whole expression will equate to $\bot$.

The bunch properties most used in this paper are given in the following summary. E,F and G are bunches and a is an element. Their types, where required, are given by context.

Bunch union

$$E, F = F, E \qquad (E, F), G = E, (F, G) \qquad E, \mathbf{null} = E$$
$$E, \bot = \bot$$

Lifting

$$E + (F, G) = E + F, E + G \qquad E \mapsto (F, G) = E \mapsto F, E \mapsto G$$
$$(E, F) \mapsto G = E \mapsto G, F \mapsto G$$

Arithmetic, (in the last two of these rules E is proper)

$$E + F = F + E \qquad E * F = F * E \qquad E + \bot = \bot \qquad E * \bot = \bot$$
$$E + \mathbf{null} = \mathbf{null} \qquad E * \mathbf{null} = \mathbf{null}$$

Distributivity

$$g \longrightarrow (E, F) = g \longrightarrow E, g \longrightarrow F$$
$$a * (E + F) = a * E + a * F \ \ (a \text{ an element})$$

2.2 Precedence and Parsing

Precedence of infix symbols, in descending order, with those of equal priority listed within brackets, is $_o\ (*\ /)\ (+\ -)\ \ _p+\ \times\ ^\wedge\ \cap\ \cup\ \setminus\ \mapsto\ \longrightarrow\ |\ ,\ (<\ >\ \leq\ \geq\ )\ (=\ \in\ :\ \neq\ \notin)\ \neg\ \wedge\ \vee\ \Rightarrow\ \Leftrightarrow\ :=\ \sqcap\ \ _p\oplus\ \sqcup\ \Longrightarrow\ \vdash\ ;\ .\ \bullet\ \diamond\ (\widehat{=}\ =\ \equiv\ \Rrightarrow\ \Lleftarrow)$. Unary symbols have higher precedence than related infix symbols, e.g logical not $\neg$ binds more tightly than the logical infix connectives.

We make a syntactic distinction between terms representing declarations, values, predicates and programs. Precedence is governed by well-formedness. For example in the expression

$$x = 1 \wedge y = 2 \longrightarrow 0$$

the highest-priority connective is $\longrightarrow$ but $2 \longrightarrow 0$ is ill-formed and this reduction is rejected. The connective of next highest priority is $=$, and the first reduction is to $(x = 1) \wedge y = 2 \longrightarrow 0$. The next is to $(x = 1) \wedge (y = 2) \longrightarrow 0$. The guard symbol now has a predicate to its left and a value to its right, suggesting a possible reduction to $(x = 1) \wedge ((y = 2) \longrightarrow 0)$, but this is rejected because the result is ill formed, having an $\wedge$ symbol between a predicate and a value. The final reduction is thus to $((x = 1) \wedge (y = 2)) \longrightarrow 0$.

3 Reversibility, Non-determinism and Backtracking

Discussion of "non-deterministic choice" (which, of course, we distinguish from "probabilistic choice") goes back at least as far as Floyd's 1967 paper "Non-deterministic Algorithms" [Flo67], where it is used to indicate provisional choices

made during a search. Within the formal methods community, the use of non-determinism as an essential abstraction tool (demonic or implementor's choice) has aroused more interest. However, as a number of writers have commented, a single semantics of non-deterministic choice can serve both purposes [Mor88, Nel89, Heh93].

In previous work, we have explored the use of non-deterministic choice used as provisional choice within search procedures, both in the B formalism [ZSD05] and in terms of Hoare and He's unifying theories [SZL06]. We propose the formalism $D \diamond E$ to represent the value(s) E might take after executing D. By adding probabilistic choice to our language we will obtain an interpretation of the expectation of $D \diamond E$ as the expected value(s) of E after conducting "experiment" D.

Central to our project is the provision of an execution platform for the constructs we investigate in the form of the Reversible Virtual Machine [Sto06]. We have created this platform in order to experiment with computations which are organised in a way that minimises essential power requirements, as analysed by Landauer [Lan61] and Feynman [Fey96]. This analysis maintains that heat is necessarily generated within a computation only where information is erased. We organise our computations in a similar way to Bennett [Ben73, Ben82] and Zuliani [Zul01], providing a history stack to preserve information and recovering the space thus consumed by reverse execution after the result of a computation has been generated. Our approach has the original aspect of using reversibility to support backtracking, and does so without compromising stepwise reversibility.

For the high level language which runs on the RVM we propose an extended expression language in which we can use terms of the form $D \diamond E$. This yields the value (or bunch of values) E would take after executing D, but *does not change the system's state*. Operationally it represents the execution of D, the recording of the value of E, then reverse execution which will return to the most recent choice construct and look for an unexplored alternative. If such an alternative is found, forward execution re-commences, and a new value is added to the bunch of results. Otherwise execution continues in reverse. On termination of the evaluation of $D \diamond E$ the original system state has been restored. For a more extensive overview see our UTP symposium paper [SZL06].

We do not obtain any benefits in terms of power consumption from our reversible virtual machine, since that just simulates reversibility on a conventional architecture. There is, however, a second important advantage of reversibility which we do exploit: reversibility provides a simple and efficient form of automatic garbage collection. This makes it relatively easy to use mathematically oriented data structures built on sets, and the RVM includes a complete implementation of finite sets and set operations.

To control reversibility we introduce naked guarded command of the form $g \Longrightarrow D$. If such a command is invoked where g is false it causes reverse execution. Otherwise the commands behaves as D. The introduction of such commands requires the repeal of Dijkstra's "Law of the Excluded Miracle". In UTP this means suspending healthiness condition **H4**, which insists that designs should be feasible.

In the following discussions $D \mathrel{\widehat{=}} P \vdash Q$ will denote a design with assumption P and commitment Q; we refer to it generally as D, but as $P \vdash Q$ when we need to explicitly mention its assumption or commitment. We define:

$$g \Longrightarrow (P \vdash Q) \;\widehat{=}\; (g \Rightarrow P \vdash g \wedge Q)$$

Note that this is a design, but not one that will obey **H4**, other than in the trivial case where $g = \mathbf{true}$.

One effect of introducing this definition is that the conditional $D_1 \lhd b \rhd D_2$ is no longer a primitive construct: it can be defined in terms of guard and non-deterministic choice as:

$$D_1 \lhd b \rhd D_2 \;\widehat{=}\; b \Longrightarrow D_1 \sqcap \neg b \Longrightarrow D_2$$

Let $P \vdash Q$ be a design with state variable (or variable list) $v : V$. Then we define:

$$(P \vdash Q) \diamond E \;\widehat{=}\; P \mid\mid \S\, v' \bullet Q \longrightarrow E[v'/v]$$

We can then prove [SZL06] the following rules, which give the effect of $D \diamond E$ over the fundamental syntactic constructs[1] of our language. Each rule eliminates one program connective, and they provide a complete characterisation. In these rules x may be an atomic variable or a variable list.

Name	Rule	Side Cond's
Assumption	$(P \vdash D) \diamond E \;=\; P \mid\mid D \diamond E$	
Skip	$I\!I \diamond E \;=\; E$	
Assignment	$x := F \diamond E \;=\; E[F/x]$	
Guard	$g \Longrightarrow D \diamond E \;=\; g \longrightarrow D \diamond E$	
Choice	$D_1 \sqcap D_2 \diamond E \;=\; (D_1 \diamond E), (D_2 \diamond E)$	
Choice from Set	$(x :\in A) \diamond E \;=\; \S\, a \bullet a \in A \longrightarrow E[a/x]$	$a \setminus E,\;\; a \setminus A$
Sequential Composition	$D_1 \,;\, D_2 \diamond E \;=\; D_1 \diamond D_2 \diamond E$	
Local Variable	$var\ z \;.\; D \diamond E \;=\; \S\, z \bullet D \diamond E$	

These rules provide an alternative semantics for sequential programs, which we refer to as "prospective-value semantics". A simple design which we will use to illustrate their use is

$$D \;\widehat{=}\; x := 1 \;\sqcap\; x := 2 \;;\; x = 2 \Longrightarrow I\!I$$

and we will calculate the possible values of x after running D. We note from our rules that the effect of a non-deterministic choice is captured by a bunch union. The first operation of D is a choice between assigning $x := 1$ or $x := 2$. However, if the choice $x := 1$ is taken, the following command becomes infeasible. In this case our rules will ensure that this blocked path through the computation makes a **null** contribution to the final result. If the choice $x := 2$ is taken the second

[1] Examples of constructs which we do not consider as fundamental are selection statements, which as we have seen can be constructed as a choice of guarded commands, and iteration constructs, which are formulated as solutions of fixed-point equations.

command reduces to $I\!I$, the program terminates with $x = 2$ and the contribution to the final result from this path through the program is 2.

We calculate the possible values of x after running D as follows:

$$
\begin{array}{ll}
& D \diamond x \\
= & \text{“Defn of D”} \\
& x := 1 \sqcap x := 2 \,;\, x = 2 \Longrightarrow I\!I \diamond x \\
= & \text{“Sequential Composition”} \\
& x := 1 \sqcap x := 2 \diamond x = 2 \Longrightarrow I\!I \diamond x \\
= & \text{“Guard and Skip”} \\
& x := 1 \sqcap x := 2 \diamond x = 2 \longrightarrow x \\
= & \text{“Choice”} \\
& (x := 1 \diamond x = 2 \longrightarrow x), (x := 2 \diamond x = 2 \longrightarrow x) \\
= & \text{“Assignment”} \\
& 1 = 2 \longrightarrow 1, 2 = 2 \longrightarrow 2 \\
= & \text{“Guarded bunch”} \\
& \mathbf{null}, 2 \;=\; 2
\end{array}
$$

In effect, the non-deterministic choice is controlled by a guard in the *following* operation. This is the basis of our backtracking semantics. An operational interpretation would be that if the choice $x := 1$ is initially made the guard of the following operation is false, provoking reverse execution. The alternative $x := 2$ is then selected, resulting in a true guard and termination with $x = 2$. We do not see the details of the operational interpretation when following the formal analysis of course. What we see instead is that a blocked path makes no contribution to the result.

One price we pay for exploiting the backtracking aspect of guards and choice is that, if we wish to avoid refining away all the behaviour of a specification, we must take more care with the management of refinement. In the classical dispensation, we are entitled to refine $x := 1 \sqcap x := 2$ by $x := 1$, but doing that in the design D above would result in a loss of feasibility as the result would simply be **magic**, a valid refinement, but not a useful one. We therefore need to identify choice used as provisional choice and not refine it away [ZSD03, Zey07].

Prospective-value semantics provides an alternative description of sequential programs in a total-correctness setting, with exactly the same expressive power as Hoare-He designs or the weakest-precondition calculus. We have already defined the value of $D \diamond E$ in terms of the assumption P and commitment Q of a design. We can similarly recover the P and Q of a design D using the following prospective-value formulations

$$
\begin{array}{lcl}
P & \equiv & \neg\, (\bot : D \diamond \mathbf{null}) \\
Q & \equiv & x' : D \diamond x
\end{array}
$$

The contribution of bunch theory to our prospective-value semantics can be seen in the simplicity of the rules for skip and for sequential composition. These rules are given in the following table, along with the rules that would be

required for a formulation in which $D \diamond E$ represents the *set* of possible values that could be taken by E after executing D.

Name	Bunch Rule	Set Rule
Skip	$I\!I \diamond E = E$	$I\!I \diamond E = \{E\}$
Sequential Composition	$D_1 ; D_2 \diamond E = D_1 \diamond D_2 \diamond E$	$D_1 ; D_2 \diamond E = \bigcup D_1 \diamond D_2 \diamond E$

We see in the set-based rule for skip that we lose homogeneity between E and $D \diamond E$. We see a consequence of this in the set-based rule for sequential composition, where a generalised union operator must be used to flatten a set of sets.

4 Probabilistic Choice and Expected Values

To extend our theory of designs to include probabilistic designs we add a probabilistic choice operator, $D_1 \ {}_p\!\oplus\ D_2$, which chooses D_1 with probability p and D_2 with probability $1-p$. Were we to formalise our theory of probabilistic designs in terms of before-after predicates, we would require non-homogeneous predicates Q which relate *before states* to *after distributions*. For example, given:

$$D \ \widehat{=}\ x := x + 1 \ {}_{\frac{1}{3}}\!\oplus\ x := x + 2$$

the predicate describing the commitment of the design could be

$$\Delta'_x = \{x + 1 \mapsto \tfrac{1}{3}, x + 2 \mapsto \tfrac{2}{3}\}$$

Rather than pursue this approach, however, we will formulate an expectation calculus. As in classical probability theory, our expectations will range over real values or vectors of real values. We represent such vectors as sequences, and assume operations for vector addition and multiplication of vectors by scalars. We emphasise at the outset that the restriction to real values will not imply any loss of generality. Indeed, we will be able to use our expectation calculus to recover the after-state distributions associated with a probabilistic design.

Before we formulate our notion of expectation, we briefly review the concepts of random variables and expectation as they have been developed in classical probability theory.

In probability theory a discrete sample space S is a countable set and a distribution **prob** over that sample space is a function from S to the closed real interval $[0, 1]$ such that the range elements of **prob** sum to 1, and with $\mathbf{prob}(s)$ interpreted as the probability that the result of some associated experiment will be s. A random variable X on S is a function from S to the real numbers and the expected value of X is given by

$$\mathbf{E}\,(X) \ \widehat{=}\ \textstyle\sum_{s \in S} \mathbf{prob}(s) * X(s)$$

By an expressive abuse of notation, random variables are generally treated notationally as if they are values rather than functions. For example, given a random variable X we are often interested in the related random variable

whose values are the squares of those returned by X, i.e. the random variable $\lambda\, s \bullet X(s)^2$. By the abuse of notation this random variable is referred to as X^2 and its expectation as $\mathbf{E}\,(X^2)$.

We should also note that in the notation $\mathbf{E}\,(X)$ the random variable X is privileged (in terms of visibility) over the distribution **prob**, which is understood to be given by context.

In probabilistic programs, the sample space is the set of values that can be taken by program variables, our experiments are programs, and their associated distributions can be calculated from the semantics we will give to each programming construct. An appropriate compositional notation for the expected value of *expression* X after running D could be $\mathbf{E}\,(D, \lambda\, x \bullet X)$, which would allow us to treat the expectation operator as a relation[2] acting on D and on the lambda abstraction of X. However, since we have a notation for the value of X after D, namely $D \diamond X$, we prefer the notation $\mathbf{E}\,(D \diamond X)$. Note that this is a composite notation, rather than the application of a relation $\mathbf{E}$ to a value $D \diamond X$. We define $\mathbf{E}\,(D \diamond X)$ according to the following rules:

Name	Rule	Cond
Assumption	$\mathbf{E}\,(P \vdash D \diamond X) \;=\; P \mid \mathbf{E}\,(D \diamond X)$	
Skip	$\mathbf{E}\,(I\!I \diamond X) \;=\; X$	
Assignment	$\mathbf{E}\,(x := E \diamond X) \;=\; X[E/x]$	
Guard	$\mathbf{E}\,(g \Longrightarrow D \diamond X) \;=\; g \longrightarrow \mathbf{E}\,(D \diamond X)$	
Choice	$\mathbf{E}\,(D_1 \sqcap D_2 \diamond X) \;=\; \mathbf{E}\,(D_1 \diamond X), \mathbf{E}\,(D_1 \diamond X)$	
Sequential Composition	$\mathbf{E}\,(D_1 \,;\, D_2 \diamond X) \;=\; \mathbf{E}\,(D_1 \diamond \mathbf{E}\,(D_2 \diamond X))$	
Local Variable	$\mathbf{E}\,(\mathit{var}\ z \,.\, D \diamond X) \;=\; \S\, z \bullet \mathbf{E}\,(D \diamond X)$	
Probabilistic Choice 1	$\mathbf{E}\,(D_1 \;{}_p\!\oplus\; D_2 \diamond X) \;=$ $\mathbf{E}\,(D_1 \diamond X) \;{}_p\!+\; \mathbf{E}\,(D_2 \diamond X)$	$0 < p$ < 1
Probabilistic Choice 2	$\mathbf{E}\,(D_1 \;{}_0\!\oplus\; D_2 \diamond X) \;=\; \mathbf{E}\,(D_2 \diamond X)$	
Probabilistic Choice 3	$\mathbf{E}\,(D_1 \;{}_1\!\oplus\; D_2 \diamond X) \;=\; \mathbf{E}\,(D_1 \diamond X)$	

A simple proof by structural induction will show that in the absence of probabilistic choice $\mathbf{E}\,(D \diamond E) = D \diamond E$. This suggests that the theory of designs is a sub-theory of the theory of probabilistic designs, an idea which we will make more precise in Section 6.

To model probabilistic choice we use the weighted bunch addition ${}_p\!+$ defined in the bunch section. As an (unsatisfactory) alternative we might have used the simpler rule

$$\mathbf{E}\,(S \;{}_p\!\oplus\; T \diamond E) = p * \mathbf{E}\,(S \diamond E) + (1 - p) * \mathbf{E}\,(T \diamond E)$$

[2] As discussed in Section 2.1 we allow the application of a relation to any argument in its domain and obtain the bunch of associated range elements as the result.

which would be correct in the case of feasible S and T, but would have the unwanted effect of making a possibly infeasible operation *certainly* infeasible. For example we would obtain $\mathbf{E}\,(\mathbf{false} \longrightarrow I\!I \;{}_p\!\oplus\; I\!I \diamond 1) = \mathbf{null}$ indicating that the operation has no after states. To write code that complies with such a rule we need to probe the feasibility of each of the choices, which would be very inefficient. We therefore prefer execution to resolve infeasibility by use of its backtracking mechanism, and the rule we adopt makes $\mathbf{false} \longrightarrow I\!I \;{}_p\!\oplus\; I\!I$ equal to $I\!I$. Thus, for us, **magic** is a zero element of probabilistic choice, just as it is for non-deterministic choice. Also, by our rule, **abort** dominates in probabilistic choice, just as it does for non-deterministic choice.

The probability or probabilities of postcondition Q holding after D is given by

Definition 2

$$\mathbf{prob}_D(Q) \;\widehat{=}\; \mathbf{E}\,(D \diamond [Q])$$

where $[Q]$ is a "numerotized predicate" taking the value 0 where Q is false and 1 where Q is true, and defined by

$$[Q] \;\widehat{=}\; Q \longrightarrow 1, \neg\, Q \longrightarrow 0$$

Note that $\mathbf{prob}_D(Q)$ may be a bunch of more than one element, and the smallest probability is given by $\mathbf{minset}\,\{\mathbf{prob}_D(Q)\}$. McIver and Morgan, whose approach is based on a probabilistic version of the wp calculus, call this the "weakest pre-expectation of Q". They take their minimums progressively by defining (in their notation) $wp.(D_1 \sqcap D_2).postE = wp.D_1.postE \;\mathbf{min}\; wp.D_2.postE$. That gives them an approach focused on wp analysis, since they assume that only the non-deterministic choice which gives the least chance of a postcondition being true is of any interest. We will use additional information retained by our rules in a number of ways, including extracting the distributions of a design and formulating alternative forms of probabilistic choice.

It is in handling the additional information present in our calculus that the use of bunches plays its most vital rôle. If we had formulated our expected values in terms of sets, as briefly discussed in the previous section, it would have been necessary to develop a theory of expectation based on *sets of real values* rather than real values themselves. That would not have been impossible, but would have required a number of special operation, such as a weighted addition of sets of values that has isomorphic properties to those of bunch addition. The theory would also have been unnecessarily clumsy when dealing with expectations that involve no non-determinism, whereas our approach makes a smooth transition between probabilistic designs involving non-determinism and those which do not.

We now demonstrate how we can extract the after-state distributions associated with a probabilistic design. We first consider designs whose alphabet consists of a single non-auxiliary variable[3] x ranging over values from the set $S = \{a, b\}$. Considering the example design

$$D_1(p) \;\widehat{=}\; x := a \;{}_p\!\oplus\; x := b$$

[3]The auxiliary variables of a design being ok and ok'.

we would intuitively expect it to yield an after-state distribution in which x takes the value a with probability p and b with probability $1-p$. We will represent such information in two different ways. Firstly as the sequence $\langle\, p, 1-p \,\rangle$, which we will call "a point in distribution space". Distribution space, (interpreted as Euclidean space or in the infinite case as a Hilbert space), will play an important part in our discussions, and the use of sequences to represent points will allow us to handle space with countably infinite dimensions. The second representation of distributions we will use is a mapping from values to probabilities, which in this case would be $\{a \mapsto p, b \mapsto 1-p\}$, and we call this a "probability distribution".

We begin with the following definition, which maps before states to points in distribution space.

$$\mathbf{dist}(D) \;\widehat{=}\; \lambda\, x \bullet \mathbf{E}\,(D \diamond \langle\, [x = a], [x = b] \,\rangle)$$

Now take the design D_1 given above where $0 < p < 1$. We calculate

$\mathbf{dist}(D_1(p))$

$=$ "Defn of **dist**"

$\lambda\, x \bullet \mathbf{E}\,(D_1(p) \diamond \langle\, [x = a], [x = b] \,\rangle)$

$=$ "Defn of D_1"

$\lambda\, x \bullet \mathbf{E}\,(x := a \;{}_p\!\oplus\; x := b \diamond \langle\, [x = a], [x = b] \,\rangle)$

$=$ "Probabilistic Choice with $0 < p < 1$"

$\lambda\, x \bullet \mathbf{E}\,(x := a \diamond \langle\, [x = a], [x = b] \,\rangle) \;{}_p\!+\; \mathbf{E}\,(x := b \diamond \langle\, [x = a], [x = b] \,\rangle)$

$=$ "Assignment"

$\lambda\, x \bullet \langle\, [a = a], [a = b] \,\rangle \;{}_p\!+\; \langle\, [b = a], [b = b] \,\rangle$

$=$ "Evaluation of $[\ldots]$"

$\lambda\, x \bullet \langle\, 1, 0 \,\rangle \;{}_p\!+\; \langle\, 0, 1 \,\rangle$

$=$ "Weighted addition with non-null arguments"

$\lambda\, x \bullet p * \langle\, 1, 0 \,\rangle + (1-p) * \langle\, 0, 1 \,\rangle$

$=$ "Multiplication of vectors by scalar"

$\lambda\, x \bullet \langle\, p, 0 \,\rangle + \langle\, 0, 1-p \,\rangle$

$=$ "Vector addition"

$\lambda\, x \bullet \langle\, p, 1-p \,\rangle$

In this case the result is a function which always returns the constant value $\langle\, p, 1-p \,\rangle$. This is because the before state has no effect on the after state, so the same distribution is obtained from any starting state.
We next consider a design in which D_1 appears as an element.

$$D_2 \;\widehat{=}\; x = a \Longrightarrow (D_1(p) \sqcap D_1(q)) \;\sqcap\; x = b \Longrightarrow D_1(r)$$

Once again we are interested in calculating distribution points for the after state, which will now depend on the before state. From the definition of **dist** and application of the rules for $\mathbf{E}\,(D \diamond X)$ we obtain

$$\mathbf{dist}(D_2) \;=\; \lambda\, x \bullet x = a \longrightarrow (\,\langle\, p, 1-p \,\rangle, \langle\, q, 1-q \,\rangle\,), x = b \longrightarrow \langle\, r, 1-r \,\rangle$$

To obtain the possible points in distribution space from pre-state $x = a$, we evaluate $\mathbf{dist}(D_2)(a)$, yielding the bunch of two distributions $\langle\, p, 1-p \,\rangle, \langle\, q, 1-q \,\rangle$. Similarly from pre-state $x = b$ we obtain the single distribution $\langle\, r, 1-r \,\rangle$.

We now consider how to express the effect of D_2 as a relation from states to probability distributions. We first define for sequences of elementary values $x = \langle\, x_1, x_2, \ldots \,\rangle$ and $y = \langle\, y_1, y_2, \ldots \,\rangle$ the combinator $x \;{}_o\; y \mathrel{\widehat{=}} \{x_1 \mapsto y_1, x_2 \mapsto y_2, \ldots\}$. We can then define (still within our limited example world) the probabilistic relation of a design as

$$\mathbf{prel}(D) \;\mathrel{\widehat{=}}\; \{x \mid x \in S \bullet x \mapsto \langle\, a, b \,\rangle \;{}_o\; \mathbf{dist}(D)(x)\}$$

and again we will generalise this definition presently. For D_2 we have:

$\mathbf{prel}(D_2)$

$=$ "Defn of **prel**"

$\{x \mid x \in S \bullet x \mapsto \langle\, a, b \,\rangle \;{}_o\; \mathbf{dist}(D_2)(x)\}$

$=$ "By comprehension of set terms"

$\{a \mapsto \langle\, a, b \,\rangle \;{}_o\; \mathbf{dist}(D_2)(a), b \mapsto \langle\, a, b \,\rangle \;{}_o\; \mathbf{dist}(D_2)(b)\}$

$=$ "Expanding $\mathbf{dist}(D_2)$"

$$\{a \mapsto \langle\, a, b \,\rangle \;{}_o\; (\lambda\, x \bullet x = a \longrightarrow (\,\langle\, p, 1-p \,\rangle, \langle\, q, 1-q \,\rangle\,), \; x = b \longrightarrow \langle\, r, 1-r \,\rangle)(a),$$
$$b \mapsto \langle\, a, b \,\rangle \;{}_o\; (\lambda\, x \bullet x = a \longrightarrow (\,\langle\, p, 1-p \,\rangle, \langle\, q, 1-q \,\rangle\,), \; x = b \longrightarrow \langle\, r, 1-r \,\rangle)(b)\}$$

$=$ "Function application and evaluation of guarded bunches"

$\{a \mapsto \langle\, a, b \,\rangle \;{}_o\; (\,\langle\, p, 1-p \,\rangle, \langle\, q, 1-q \,\rangle\,) \;,\; b \mapsto \langle\, a, b \,\rangle \;{}_o\; \langle\, r, 1-r \,\rangle\}$

$=$ "Bunch lifting of ${}_o$ and $\mapsto$"

$\{a \mapsto \langle\, a, b \,\rangle \;{}_o\; \langle\, p, 1-p \,\rangle, a \mapsto \langle\, a, b \,\rangle \;{}_o\; \langle\, q, 1-q \,\rangle, b \mapsto \langle\, a, b \,\rangle \;{}_o\; \langle\, r, 1-r \,\rangle\}$

$=$ "Applying the sequence combinator ${}_o$"

$\{a \mapsto \{a \mapsto p, b \mapsto 1-p\}, a \mapsto \{a \mapsto q, b \mapsto 1-q\}, b \mapsto \{a \mapsto r, b \mapsto 1-r\}\}$

We now demonstrate how such a relation from states to probability distributions can be used to calculate an expectation, under the assumption of a given pre-distribution Δ. We note that $\mathbf{prel}(D_2)$ from our above example has two subsets which have the same domain as $\mathbf{prel}(D_2)$ and which are functional, namely

$$F_1 = \{a \mapsto \{a \mapsto p, b \mapsto 1-p\}, b \mapsto \{a \mapsto r, b \mapsto 1-r\}\}$$

$$F_2 = \{a \mapsto \{a \mapsto q, b \mapsto 1-q\}, b \mapsto \{a \mapsto r, b \mapsto 1-r\}\}$$

These model deterministic (though probabilistic) operations. Note that, for example, $F_1(b)(a)$ is the probability of F_1 going from a before state b to after state a. The probability that F_1 will yield $x' = a$ is the probability of starting in a and finishing in a plus the probability of starting in b and finishing in a, that is:

$$\Delta a * F_1(a)(a) + \Delta b * F_1(b)(a) = \Delta a * p + \Delta b * r$$

Now letting $S = \{a, b\}$, the expectation of expression E after F_1 will be:

$$\textstyle\sum_{s \in S, s' \in S} \Delta s * F_1(s)(s') * E[s'/x]$$

and the expectation of E after D_2 will be the bunch of two such terms obtained from F_1 and F_2.

For a generalised model consider a design $D \mathrel{\widehat{=}} P \vdash Q$ whose alphabet consists of a variable or variable list s taking values from a countable set $S = \{s_1, s_2, \ldots\}$. Then the distributions of D are given by

Definition 3 $\mathbf{dist}(D) \;\widehat{=}\; \mathbf{E}\,(D \diamond \langle\, [s = s_1], [s = s_2], \ldots \rangle)$

and the probabilistic relation of D is given by

Definition 4 $\mathbf{prel}(D) \;\widehat{=}\; \{s \mid s \in S \bullet s \mapsto \langle\, s_1, s_2, \ldots \rangle \,{}_o\, \mathbf{dist}(D)(s)\}$

To model expectations we define a function to return the functions "packed" within a relation:

$$\mathcal{F}(R) \;\widehat{=}\; \{F \mid F \subseteq R \wedge \mathbf{dom}\ F = \mathbf{dom}\ R \wedge F \in \mathbf{dom}\ R \to \mathbf{ran}\ R\}$$

and the expectation of E after D assuming an initial distribution Δ is:

$$\S\ F \bullet F \in \mathcal{F}(\mathbf{prel}(D)) \longrightarrow \textstyle\sum_{s \in S, s' \in S} F(s)(s') * E[s'/x]$$

A relational model from states to distributions over states was first proposed by He, Seidel and McIver [HSM97] and is discussed in [MM04] though formulated as a function from states to sets of distributions.

5 Convexity and Refinement

Suppose we have designs D_1 and D_2 which act on the same state space, and suppose that, for any postcondition Q, the minimum probability of D_2 establishing Q is at least as great as the minimum probability of D_1 doing so. Then, by analogy with standard operational refinement, we might be tempted to say that D_1 is refined by D_2. Such a definition of refinement would, however, not be monotonic with respect to program connectives, as is shown on pages 314–315 of [MM04]. In this section we attempt to provide some intuitive justification for this result by means of a geometric model, and then show how non-deterministic choice used as provisional choice must be handled during the refinement process.

We consider a state space with a single variable v which can range over the distinct values a, b and c. A program acting on this state space has eight possible postconditions: **true**, **false**, $v = a$, $v = b$, $v = c$, $v \neq a$, $v \neq b$ and $v \neq c$. Consider a program which offers a non-deterministic choice between six distributions $d_1, \ldots d_6$ for assigning a value to v. In the following table we write the probabilities of each distribution yielding an a, b or c, labelling these as x, y and z to comply with a geometric interpretation in which we will represent these probabilities as points in three dimensional space. To the right of them we write lower and upper bound constraints imposed by each distribution.

	x	y	z	lower bound	upper bound
d_1	0.1	0.4	0.5	$\mathbf{prob}(v = a) \geqslant 0.1$	$\mathbf{prob}(v \neq a) \leq 0.9$
d_2	0.5	0.1	0.4	$\mathbf{prob}(v = b) \geqslant 0.1$	$\mathbf{prob}(v \neq b) \leq 0.9$
d_3	0.4	0.5	0.1	$\mathbf{prob}(v = c) \geqslant 0.1$	$\mathbf{prob}(v \neq c) \leq 0.9$
d_4	0.2	0.2	0.6	$\mathbf{prob}(v \neq c) \geqslant 0.4$	$\mathbf{prob}(v = c) \leq 0.6$
d_5	0.6	0.2	0.2	$\mathbf{prob}(v \neq a) \geqslant 0.4$	$\mathbf{prob}(v = a) \leq 0.6$
d_6	0.2	0.6	0.2	$\mathbf{prob}(v \neq b) \geqslant 0.4$	$\mathbf{prob}(v = b) \leq 0.6$

Table of Distributions.

Distribution d_1 has the lowest probability of all distributions for establishing $v = a$. It imposes the constraints that, after running our program, the probability of $v = a$ will be at least 0.1. Suppose we choose some value $p < 0.1$ and by running our program many times we collect data to test the hypothesis $\mathbf{prob}(v = a) = p$. To do this we perform trials by repeatedly running our program and recording, after each run, whether $v = a$ is true or false. If we perform a hypothesis test based on these results it will fail with a probability arbitrarily close to 1 if we base it on a sufficiently large number of trials. This is true no matter what choices the demon makes. Similarly any hypothesis that takes some $p > 0.9$ and posits $\mathbf{prob}(v \neq a) = p$ will similarly fail. The constraints shown in the table show the limits outside which such hypotheses will fail with probability arbitrarily close to 1 given a test based on a sufficiently large number of trials.

We can give a geometric illustration of these distributions and constraints. In the left-hand diagram of Fig. 1 we see a portion of 3D Euclidean space within which we have drawn the triangle with corners at $\langle 0, 0, 1\rangle, \langle 0, 1, 0\rangle, \langle 1, 0, 0\rangle$. On the plane defined by these points we have $x + y + z = 1$, and within the triangle we have $0 \leq x \leq 1 \wedge 0 \leq y \leq 1 \wedge 0 \leq z \leq 1$. Each point within the triangle corresponds to a unique distribution, with the x, y and z coordinates representing the probabilities $v = a$, $v = b$ and $v = c$.[4]

In the right-hand diagram of Fig. 1 we have taken the triangle of distributions and drawn it flat on the page. Now, along the base of the triangle we have $y = 0$ and at the top we have $y = 1$. We have drawn lines on the triangle to mark the points at which $y = 0.1$ and $y = 0.6$.

In Fig. 2 we have marked all the restrictions shown in the table of distributions along with the points corresponding to the distributions d_1 to d_6 themselves. Joining the points d_1 to d_6 as shown, we form a convex hexagon. Taking any distribution within the hexagon and testing whether our program has that distribution is *not* bound to fail in the manner previously described, because the demon could make a judicious choice between his available distributions to provide a distribution at that particular point. We call the region within the hexagon the *convex closure* of the points $d_1, \ldots d_6$. The convex closure of a set of points in Euclidean space is the smallest set containing the points themselves, and any point on a straight line between two other points in the closure.

We have also marked a question mark on the diagram of Fig. 2. This lies outside the convex closure of the distributions, yet does not conflict with the

[4] A similar geometric representation is given in [MM04], but in their case accommodates sub-distributions. Their space of sub-distributions is 3-dimensional, occupying the tetrahedron with apexes (expressed in conventional tuple notation) at $(0, 0, 0)$, $(0, 0, 1)$, $(0, 1, 0)$ and $(1, 0, 0)$.

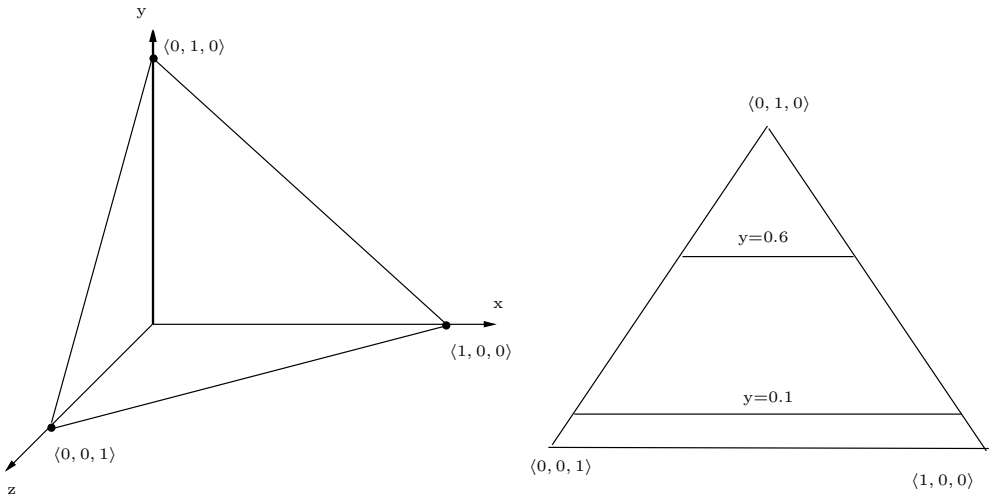

Figure 1: The space of distributions over three possible values.

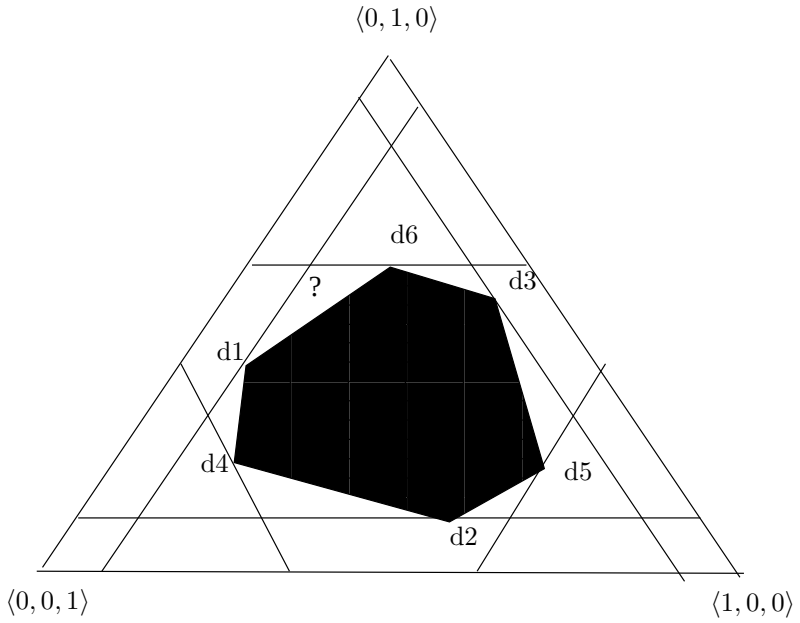

Figure 2: The convex closure and event constraints of a set of distributions.

constraints imposed by any postcondition. No test based on the observation of a particular postcondition can be sure of refuting the hypothesis that we are observing a distribution situated at this point. We *can* design an experiment which, based on a sufficient number of observations, is sure to refute this hypothesis, but we must record all the information available from each trial. If we record, after each trial, the value of $\langle [x = a], [x = b], [x = c] \rangle$, the vector average of these values will approach the given point with probability 1 over a long sequence of trials. In terms of hypothesis testing, a χ^2 test could be used to tell us the significance of a particular set of results.

If, on the other hand, we modify our program to add a seventh distribution, but we place this new distribution within the convex closure defined by the other six, there is no experiment based on observing repeated runs of the program that will be able to detect any difference made by this additional distribution. This consideration has led McIver and Morgan [MM04] to use convex closure (more specifically convex "up closure" which also deals with sub-distributions) as the key concept for defining the meaning of probabilistic programs and refinement. They identify probabilistic programs with their convex closures and define refinement as containment. We now look at the problems we face in adopting exactly this approach with our reversible language in which non-deterministic choice serves both as provisional choice in a backtracking context, and as implementor's choice.

In [ZSD03] we consider refinement in the context of reversible computation, and note the danger of over refinement in a language where our syntax allows us to express infeasible programs. For example **magic** is a design that we can express in our implementation language. It refines any program, but is unlikely to satisfy any customer. Its run-time behaviour, in an operational sense, is to put execution into reverse. We define "star-refinement" $\sqsubseteq^*$ designed to guard against the problem of over-refinement by maintaining feasibility during the refinement process. In the following definition $\mathrm{fis}(D)$ is the condition that D is feasible in the current state.

Definition 5

$$D_A \sqsubseteq^* D_C \;\widehat{=}\; D_A \sqsubseteq D_C \;\wedge\; (\mathrm{fis}(D_A) \;\Rrightarrow\; \mathrm{fis}(D_C))$$

This definition refers to non-probabilistic programs. We will use the idea of preserving feasibility during refinement to produce a similar definition for the probabilistic case.

For star-refinement to serve as a basis for piecewise and stepwise development, we would like our language constructs to be monotonic with respect to it. We prove in [ZSD03] that they are so for assumption, guard, assignment, choice and sequential composition in its second operand.

We can illustrate the case of sequential composition in its first operand with the program:

$$x := 1 \sqcap x := 2 \,;\, x = 2 \longrightarrow \mathit{II}$$

this is equivalent to $x := 2$, but if we star-refine its first operand by $x := 1$, the resulting program $x := 1 \,;\, x = 2 \longrightarrow \mathit{II}$ is equal to **magic**. It illustrates a typical backtracking situation in which the non-deterministic choice of the first operand is acting as provisional choice. An operational interpretation was given

earlier. To manage star-refinement such provisional choice must be identified and *not refined*.

To give a definition of star-refinement in the probabilistic world we first provide a definition for feasibility of a probabilistic design D with state variables s. The idea of this definition is that only infeasibility can prevent the expected value of $\bot$ after executing D being $\bot$.

Definition 6 $\mathrm{fis}(D) \;\widehat{=}\; \mathbf{E}\,(D \diamond \bot) = \bot$

The definition of star-refinement in a probabilistic world is:

Definition 7

$$\begin{array}{l} D_A \sqsubseteq^* D_C \;\widehat{=}\; (P_A \;\Rrightarrow\; P_C) \wedge \\ \quad (\forall\, s \bullet \mathbf{conv}(\{\mathbf{dist}(P_A \Longrightarrow D_C)(s)\}) \subseteq \mathbf{conv}(\{\mathbf{dist}(P_A \Longrightarrow D_A)(s)\})) \\ \qquad\qquad \wedge\; \mathrm{fis}(D_A) \;\Rrightarrow\; \mathrm{fis}(D_C) \end{array}$$

where **conv** maps a set of points in distribution space to its convex closure. We require D_C to terminate whenever D_A does, we restrict the required subset inclusion to the termination region of D_A, and we require the refinement to be no less feasible than the program it refines.

We add a final note on infinite state spaces. Euclidean space is normally thought of as having a finite number of dimensions. An infinite-dimensional space with the Euclidean metric loses the important property that the distance between any two points in the space is well-defined. If we take an n dimensional cube of fixed size δ (however small), the length of the longest diagonal between vertices of this cube will be $\delta * \sqrt{n}$, a term which tends to infinity with n. Thus in infinite-dimensional space, we have infinite distances within arbitrarily small hypercubes. However, within distribution space, which is the space of sequences whose elements represent probabilities, we have the following:

Proposition 1 *The Euclidean metric is defined between any two points in infinitely dimensional distribution space and the square of the distance between such points, say* $\langle\, x_1, x_2 .. \,\rangle$ *and* $\langle\, y_1, y_2 .. \,\rangle$*, which by the Euclidean metric is* $\Sigma_{i=1}^{\infty}(x_i - y_i)^2$*, is less than or equal to 2.*

Proof

$\sum_{i=1}^{\infty}(x_i - y_i)^2$

= "Expanding each term"

$\sum_{i=1}^{\infty}(x_i^2 - 2x_i y_i + y_i^2)$

$\leq$ "Since $2x_i y_i > 0$"

$\sum_{i=1}^{\infty}(x_i^2 + y_i^2)$

= "By associativity of addition"

$\sum_{i=1}^{\infty} x_i^2 + \sum_{i=1}^{\infty} y_i^2$

$\leq$ "Since $0 \leq x_i \leq 1 \wedge 0 \leq y_i \leq 1$"

$\sum_{i=1}^{\infty} x_i + \sum_{i=1}^{\infty} y_i$

= "Since probabilities sum to 1"

2

□

6 Linking Theories

We can link our probabilistic designs to classical designs by means of a Galois connection.

Definition 8 *Given a pair of partially ordered sets (posets) A and B, and functions*

$$L \in A \to B,\ R \in B \to A$$

then L and R form a Galois connection if

$$\beta \sqsubseteq L(\alpha) \ \Leftrightarrow\ R(\beta) \sqsubseteq \alpha \tag{1}$$

L will play the rôle of a transformation from the richer world of probabilistic designs to the world of designs, and it will lose information in a controlled way by treating probabilistic choice as non-deterministic choice. The function R will lift a design into the world of probabilistic designs by expressing values as point distributions, i.e. distributions which give a particular value with probability one. The existence of a Galois connection will tell us that should we perform the transformation L from a probabilistic design to a design and then perform the transformation R to take the result back to the probabilistic world, the result will be an abstraction (anti-refinement) of our original probabilistic design. Performing transformations in the opposite direction, we will be able to lift a design into the probabilistic world and transfer it back again to the same design, a stronger property than required for a Galois connection, and the characteristic of a "retract".

Before applying these ideas to our theory, we give an equivalent characterisation of Galois connections that is more directly related to the transformations we have just discussed.

Proposition 2 *Given posets A, B, a pair of monotonic functions $L \in A \to B$ and $R \in B \to A$ form a Galois connection (L, R) if*

$$R(L(\alpha)) \sqsubseteq \alpha \tag{2}$$

$$\beta \sqsubseteq L(R(\beta)) \tag{3}$$

We prove the given conditions are sufficient for (L, R) to be a Galois connection. This will be enough for our purposes, though it can also be readily shown that the conditions are also necessary.

ProofWe need to show from the given conditions that (1) holds. We first assume the LHS of (1), that is $\beta \sqsubseteq L(\alpha)$ and prove the RHS

$\beta \sqsubseteq L(\alpha)$

$\Rightarrow$ "by monotonicity of R"

$R(\beta) \sqsubseteq R(L(\alpha))$

$\Rightarrow$ "by (2) and transitivity of $\sqsubseteq$"

$R(\beta) \sqsubseteq \alpha$

And now we assume the RHS of (1) and prove the LHS

$R(\beta) \sqsubseteq \alpha$

$\Rightarrow$ "by monotonicity of L"

$L(R(\beta)) \sqsubseteq L(\alpha)$

$\Rightarrow$ "by (3) and transitivity of $\sqsubseteq$"

$\beta \sqsubseteq L(\alpha)$ $\Box$

In the following discussion $\mathcal{D}$ and $\mathcal{D}'$ will represent probabilistic designs and D and D' classical designs. To characterise the Galois connection between designs and probabilistic designs we use the relational models of each. For a design D acting on a state variable $s \in S$ we define

Definition 9

$$\mathbf{rel}(D) \;\widehat{=}\; \{s \mid s \in S \bullet s \mapsto D \diamond s\}$$

and for a probabilistic design we use the definition of **prel** previously given in Definition 4.

As an example we take the design acting on variable $v \in \{a, b, c\}$ defined by:

$$\begin{array}{l} \mathcal{D} \;\widehat{=}\; \\ \quad v = a \Longrightarrow ((v := a \;{}_{\frac{1}{2}}\!\oplus\; v := c) \sqcap (v := b \;{}_{\frac{1}{2}}\!\oplus\; v := c) \sqcap \mathit{II}) \\ \qquad \sqcap \\ \quad v = b \Longrightarrow \\ \qquad ((v := a \;{}_{\frac{1}{3}}\!\oplus\; v := b) \sqcap (v := a \;{}_{\frac{1}{3}}\!\oplus\; v := c) \sqcap (v := b \;{}_{\frac{1}{3}}\!\oplus\; v := c)) \end{array}$$

calculating $\mathbf{dist}(\mathcal{D}) = \mathbf{E}\,(\mathcal{D} \diamond \langle [v = a], [v = b][v = c] \rangle)$ we obtain:

$$\{a \mapsto (\langle \tfrac{1}{2}, 0, \tfrac{1}{2} \rangle, \langle 0, \tfrac{1}{2}, \tfrac{1}{2} \rangle, \langle 1, 0, 0 \rangle),\ b \mapsto (\langle \tfrac{1}{3}, \tfrac{2}{3}, 0 \rangle, \langle \tfrac{1}{3}, 0, \tfrac{2}{3} \rangle, \langle 0, \tfrac{1}{3}, \tfrac{2}{3} \rangle)\}$$

where we are using the distributivity of maplet construction over bunch union to write the contents of the set in a more compact form. The convex closures of $\{\mathbf{dist}(\mathcal{D})(a)\}$ and $\{\mathbf{dist}(\mathcal{D})(b)\}$ are shown in Fig. 3. The value of $\mathbf{prel}(\mathcal{D})$ is

$$\begin{array}{l} \{a \mapsto (\{a \mapsto \frac{1}{2}, b \mapsto 0, c \mapsto \frac{1}{2}\}, \{a \mapsto 0, b \mapsto \frac{1}{2}, c \mapsto \frac{1}{2}\}, \{a \mapsto 1, b \mapsto 0, c \mapsto 0\}) \\ b \mapsto (\{a \mapsto \frac{1}{3}, b \mapsto \frac{2}{3}, c \mapsto 0\}, \{a \mapsto \frac{1}{3}, b \mapsto 0, c \mapsto \frac{2}{3}\}, \{a \mapsto 0, b \mapsto \frac{1}{3}, c \mapsto \frac{2}{3}\})\} \end{array}$$

We define the transformation L, applicable to any probabilistic design $\mathcal{D}$ by

Definition 10

$$L(\mathcal{D}) \;\widehat{=}\; \mu\, D \bullet s \mapsto s' \in \mathbf{rel}(D) \Leftrightarrow \exists\, p \bullet s \mapsto p \in \mathbf{prel}(\mathcal{D}) \wedge p(s') > 0$$

which for our example yields

$$\mathbf{rel}(D) = \{(a, b) \mapsto (a, b, c)\}$$

which is a set of six elements expressed in more compact form using distributivity of maplet construction over bunch union. From state $v = a$ we can obtain after states of $v = a$ or $v = b$ or $v = c$, as we can from state $v = b$. From state $v = c$ D is infeasible. The design D is

$$D \;\widehat{=}\; v \in \{a, b\} \Longrightarrow (v := a \sqcap v := b \sqcap v := c) \tag{4}$$

We define the transformation R, applicable to any design D by

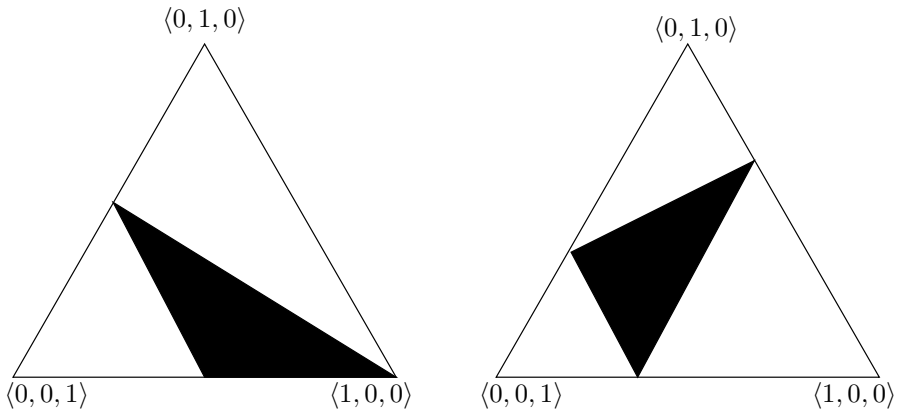

Figure 3: The convex closures of $\{\mathbf{dist}(\mathcal{D})(a)\}$ and $\{\mathbf{dist}(\mathcal{D})(b)\}$

Definition 11

$$R(D) \;\widehat{=}\; \mu\,\mathcal{D}' \bullet s \mapsto p \in \mathbf{prel}(\mathcal{D}') \Leftrightarrow \exists\, s' \bullet s \mapsto s' \in \mathbf{rel}(D) \wedge p(s') = 1$$

which for our example yields

$$\mathbf{prel}(\mathcal{D}') = \{(a, b) \mapsto \\ (\{a \mapsto 1, b \mapsto 0, c \mapsto 0\}, \{a \mapsto 0, b \mapsto 1, c \mapsto 0\}, \{a \mapsto 0, b \mapsto 0, c \mapsto 1\})\} \quad (5)$$

and the definition of $\mathcal{D}'$ would be exactly that given for D in equation 4 above, the difference being that we are now interpreting this definition in terms of our probabilistic model. The value of $\mathbf{dist}(\mathcal{D}')$ is

$$\{(a, b) \mapsto (\langle\, 1, 0, 0\,\rangle, \langle\, 0.1.0\,\rangle, \langle\, 0, 0, 1\,\rangle\}$$

We now check the Galois connection requirement $R(L(\mathcal{D})) \sqsubseteq^* \mathcal{D}$, i.e. $\mathcal{D}' \sqsubseteq^* \mathcal{D}$. The assumption and feasibility of $\mathcal{D}'$ and $\mathcal{D}$ are **true** so we need consider only the containment of convex closures, i.e.

$$\forall\, s \bullet \mathbf{conv}(\{\mathbf{dist}(\mathcal{D})(s)\}) \subseteq \mathbf{conv}(\{\mathbf{dist}(\mathcal{D}')(s)\})$$

we have $\{\mathbf{dist}(\mathcal{D}')(a)\} = \{\langle\, 1, 0, 0\,\rangle, \langle\, 0.1.0\,\rangle, \langle\, 0, 0, 1\,\rangle\}$, i.e. the point distributions which form the vertices of distribution space. Their convex closure is the whole of distribution space and must therefore contain the convex closure of $\{\mathbf{dist}(\mathcal{D})(a)\}$. Similar remarks apply to b, whereas for c the containment is satisfied because the convex closures are empty, both operations being infeasible in this case.

We argue the general case that $R(L(\mathcal{D})) \sqsubseteq^* \mathcal{D}$ informally. The transformations L and R do not alter the assumption or feasibility of a design, so we are concerned only with the containment of convex closures of distributions. We recall that each dimension of distribution space corresponds to a particular value within the state space. We can eliminate dimensions of distribution space associated with impossible after state values. For the remaining dimensions and any before state s, the convex closure of $\{\mathbf{dist}(\mathcal{D}')(s)\}$ will be the whole of

the remaining distribution space, and must therefore contain the corresponding convex closure of $\{\mathbf{dist}(\mathcal{D})(s)\}$

Now consider the second Galois connection requirement $D \sqsubseteq^* L(R(D))$. We take D as defined in equation 4, and for $\mathcal{D}' = R(D)$ we have $\mathbf{prel}(\mathcal{D}')$ as given by equation 5. Letting $D' = L(\mathcal{D}')$ we have from Definition 10 that

$$\mathbf{rel}(D') = \{(a, b) \mapsto (a, b, c)\}$$

and thus $D' = D$, i.e. $L(R(D)) = D$. Again, we argue informally, this result also holds in general since the effect of R is to lift each after state of D into a point distribution, and the effect of L is to return this point distribution to its associated value.

The importance of the retract is that it confirms in a general sense that when reasoning about the prospective-value effects of a design we can reason about probabilistic choice as though it were non-deterministic choice, and that, in so doing, we will be reasoning about an abstraction of our original probabilistic design. This, indeed, is the impact of the property $R(L(\mathcal{D})) \sqsubseteq^* \mathcal{D}$. We can also say, from the property $L(R(D)) = D$ that the theory of designs is a sub-theory of probabilistic designs, in that any reasoning about the prospective-value effect of probabilistic designs that do not involve probabilistic choice will correspond with reasoning about the effect of corresponding classical designs.

7 Interactions between Demonic and Probabilistic Choice

Suppose we have designs $D_1 \mathrel{\widehat{=}} x := a \sqcap x := b$ and $D_2 \mathrel{\widehat{=}} y := c \;{}_p\!\oplus\; y := d$, where a, b, c and d are constants. At first glance, D_1 and D_2 appear to be independent in terms of their effects, and we might suppose that $D_1 \,;\, D_2 = D_2 \,;\, D_1$. However, as both McIver and Morgan [MM04] and He and Sanders [HS06] have noted, this is not the case, and instead we have $D_2;\ D_1 \sqsubseteq^* D_1;\ D_2$. In effect, to reflect implementor's choice we need to characterise $D_2 \,;\, D_1$ (where the implementor's choice follows the probabilistic assignment) so that the implementor's choice can be made in a manner that is dependent on the current state. We will see how this works in our formalism in a moment. Then we will consider why we might want, and how we can formulate, rules to give "oblivious nondeterminism", a blind form of interaction.

To show $D_1 \,;\, D_2 \neq D_2 \,;\, D_1$ we order the possible values of $x \mapsto y$ as $a \mapsto c$, $a \mapsto d$, $b \mapsto c$, $b \mapsto d$ and calculate the distributions of $x \mapsto y$ after $D_1 \,;\, D_2$ and $D_2 \,;\, D_1$ respectively. For reasons of space we limit ourselves to the case where a, b, c and d have different values. For $D_1 \,;\, D_2$ we then have:

$\mathbf{dist}(D_1 \,;\, D_2)$

$=$ "Defn of **dist** (Def. 3)"

$\lambda\, x, y \bullet \mathbf{E}\,(D_1 \,;\, D_2 \diamond \langle\, [x \mapsto y = a \mapsto c)], [x \mapsto y = a \mapsto d], [x \mapsto y = b \mapsto c], [x \mapsto y = b \mapsto d]\,\rangle)$

$=$ "Defn of D_2, Sequential composition, Probabilistic Choice and Assignment"

$\lambda\, x, y \bullet \mathbf{E}\,(D_1 \diamond p * \langle\, [x \mapsto c = a \mapsto c], [x \mapsto c = a \mapsto d], [x \mapsto c = b \mapsto c], [x \mapsto c = b \mapsto d]\,\rangle +$

$(1-p) * \langle\, [x \mapsto d = a \mapsto c], [x \mapsto d = a \mapsto d], [x \mapsto d = b \mapsto c], [x \mapsto d = b \mapsto d)] \,\rangle)$

$=$ "Evaluation of numerotized predicates and simplification of terms"

$\lambda\, x, y \bullet \mathbf{E}\,(D_1 \diamond p * \langle\, [x = a], 0, [x = b], 0 \,\rangle + (1-p) * \langle\, 0, [x = a], 0, [x = b] \,\rangle)$

$=$ "Defn of D_1 and Choice"

$\lambda\, x, y \bullet p * \langle\, 1, 0, 0, 0 \,\rangle + (1-p) * \langle\, 0, 1, 0, 0 \,\rangle, p * \langle\, 0, 0, 1, 0 \,\rangle + (1-p) * \langle\, 0, 0, 0, 1 \,\rangle$

$=$ "Vector multiplication and addition"

$\lambda\, x, y \bullet \langle\, p, 1-p, 0, 0 \,\rangle, \langle\, 0, 0, p, 1-p \,\rangle$

The two distributions we see in this result correspond to the non-deterministic choice in which the demon has assigned $x := a$ and $x := b$ respectively. We next evaluate the distributions of $D_2\,;\, D_1$.

$\mathbf{dist}(D_2\,;\, D_1)$

$=$ "Defn of **dist** (Def. 3)"

$\lambda\, x, y \bullet \mathbf{E}\,(D_2\,;\, D_1 \diamond \langle\, [x \mapsto y = a \mapsto c], [x \mapsto y = a \mapsto d], [x \mapsto y = b \mapsto c], [x \mapsto y = b \mapsto d] \,\rangle)$

$=$ "Defn of D_1, Sequential Composition, Choice and Assignment"

$\lambda\, x, y \bullet \mathbf{E}\,(D_2 \diamond$

$\langle\, [a \mapsto y = a \mapsto c], [a \mapsto y = a \mapsto d], [a \mapsto y = b \mapsto c], [a \mapsto y = b \mapsto d] \,\rangle,$

$\langle\, [b \mapsto y = a \mapsto c], [b \mapsto y = a \mapsto d], [b \mapsto y = b \mapsto c], [b \mapsto y = b \mapsto d] \,\rangle)$

$=$ "Evaluation of numerotized predicates and Simplification of terms"

$\lambda\, x, y \bullet \mathbf{E}\,(D_2 \diamond \langle\, [y = c], [y = d], 0, 0 \,\rangle, \langle\, 0, 0, [y = c], [y = d] \,\rangle)$

$=$ "Defn of D_2, Probabilistic Choice and evaluation of numerotized predicates"

$\lambda\, x, y \bullet p * (\, \langle\, 1, 0, 0, 0 \,\rangle, \langle\, 0, 0, 1, 0 \,\rangle\,) + (1-p) * (\, \langle\, 0, 1, 0, 0 \,\rangle, \langle\, 0, 0, 0, 1 \,\rangle\,)$

$=$ "Lifted vector multiplication"

$\lambda\, x, y \bullet (\, \langle\, p, 0, 0, 0 \,\rangle, \langle\, 0, 0, p, 0 \,\rangle\,) + (\, \langle\, 0, 1-p, 0, 0 \,\rangle, \langle\, 0, 0, 0, 1-p \,\rangle\,)$

$=$ "Lifted vector addition"

$\lambda\, x, y \bullet (\, \langle\, p, 1-p, 0, 0 \,\rangle, \langle\, p, 0, 0, 1-p \,\rangle, \langle\, 0, (1-p), p, 0 \,\rangle, \langle\, 0, 0, p, 1-p \,\rangle\,)$

Here, where demonic choice follows probabilistic choice, we obtain a lambda expression whose body is a bunch of four elements. The first and fourth are those obtained from our analysis of $D_1\,;\, D_2$, and correspond to the demon assigning $x := a$ and $x := b$ respectively. The second and third elements, which we refer to as "hybrid terms", correspond to demonic choices made according to the current state. For example the element $\langle\, p, 0, 0, 1-p \,\rangle$ corresponds to the demon assigning $x := a$ when $y = c$ and $x := b$ when $y = d$. We can see that we need this element in our result if we recall that an operation is the demonic choice of all its operational refinements, and:

$$x := a \sqcap x := b \quad \sqsubseteq \quad y = c \Longrightarrow x := a \;\sqcap\; y \neq c \Longrightarrow x := b$$

We also note that the new elements are not in the convex closure of those found previously, so that we have $D_2 \,;\, D_1 \sqsubseteq^* D_1 \,;\, D_2$ but not vice versa.

McIver and Morgan [MM04] discuss a number of alternatives to their main expectation calculus, one of which is a programming logic of distributions. This is based on Hoare triples, with both the pre and post-judgements being over distributions. This formalism exhibits some different properties, one of which is that non-deterministic choice becomes blind with respect to the current state. "When we lift the whole semantic structure up to distributions, from states, the demonic choice loses the ability to see individual states: it can only see distributions.. There are circumstances in which such *oblivious non-determinism* is the behaviour we are trying to capture, for example when we are dealing with concurrency or modularity in which separation of processes, or information hiding, we protect part of the state from being read freely by other parts of the system".

To approach the formulation of oblivious choice we consider the interaction of probabilistic and demonic choice illustrated above and note that the interaction between the two forms of choice comes from hybrid terms, i.e. the addition of terms which originally arose from two different non-deterministic choices. To eliminate them we replace the probabilistic choice in our language with an alternative we refer to as "random choice", where the random choice between D_1 and D_2 which chooses D_1 with probability p and D_2 with probability $1-p$ will be represented by $D_1 \;{}_p\!\boxplus\; D_2$.

Definition 12 *For E being elementary,* **null** *or $\bot$ random choice is identical to probabilistic choice:*

$$\mathbf{E}\,(D_1 \;{}_p\!\boxplus\; D_2 \diamond E) \;\;\widehat{=}\;\; \mathbf{E}\,(D_1 \;{}_p\!\oplus\; D_2 \diamond E) \quad \textit{for} \quad \mathbf{card}(E) \leqslant 1 \textit{ or } E = \bot$$

Otherwise we demand that random choice distributes through probabilistic choice, namely for any bunches E and F we have

$$\mathbf{E}\,(D_1 \;{}_p\!\boxplus\; D_2 \diamond E, F) \;\;\widehat{=}\;\; \mathbf{E}\,(D_1 \;{}_p\!\boxplus\; D_2 \diamond E), \mathbf{E}\,(D_1 \;{}_p\!\boxplus\; D_2 \diamond F)$$

If we now take $D_1 \mathrel{\widehat{=}} x := a \sqcap x := b$ and $D_2 \mathrel{\widehat{=}} y := c \;{}_p\!\boxplus\; y := d$, we can use the method of extracting the distributions of $D_1;\; D_2$ and $D_2;\; D_1$ to show $D_1;\; D_2 \;=\; D_2;\; D_1$; the non-deterministic choice in D_1 has become oblivious of state.

It may seem strange that we obtain oblivious non-deterministic choice by changing our conception of how probabilistic choices are made, and we will investigate this further. We now consider what we call "random designs" which may be written with the connectives of classical designs plus random choice. They exclude any use of the probabilistic choice operator $\;{}_p\!\oplus$.

Proposition 3 *For any random design D, prospective value calculations distribute through bunch union, that is:*

$$\mathbf{E}\,(D \diamond E, F) \;=\; \mathbf{E}\,(D \diamond E), \mathbf{E}\,(D \diamond F)$$

Proof

Proof is by structural induction over the constructs of random designs. We consider just some constructs to show the general approach. We also omit

special case analysis for E or F being $\perp$ or **null**. As usual, assignment and skip provide base cases. For skip we have

$$
\begin{array}{ll}
 & \mathbf{E}\,(I\!I \diamond E, F) \\
= & \text{“Rule for skip”} \\
 & E, F \\
= & \text{“Rule for skip applied separately to } E \text{ and } F\text{”} \\
 & \mathbf{E}\,(I\!I \diamond E), \mathbf{E}\,(I\!I \diamond F)
\end{array}
$$

As examples of constructs which appeal to the inductive case we consider first the guard construct

$$
\begin{array}{ll}
 & \mathbf{E}\,(g \Longrightarrow D \diamond E, F) \\
= & \text{“Rule for guard”} \\
 & g \longrightarrow \mathbf{E}\,(D \diamond E, F) \\
= & \text{“inductive case”} \\
 & g \longrightarrow (\mathbf{E}\,(D \diamond E), \mathbf{E}\,(D \diamond F)) \\
= & \text{“Distributivity of bunch guards, } g \longrightarrow (E, F) \;=\; g \longrightarrow E, g \longrightarrow F\text{”} \\
 & g \longrightarrow \mathbf{E}\,(D \diamond E), g \longrightarrow \mathbf{E}\,(D \diamond F)
\end{array}
$$

For sequential composition we have

$$
\begin{array}{ll}
 & \mathbf{E}\,(D_1;\; D_2 \diamond E, F) \\
= & \text{“Rule for sequential composition”} \\
 & \mathbf{E}\,(D_1 \diamond \mathbf{E}\,(D_2 \diamond E, F)) \\
= & \text{“Inductive case on } D_2\text{”} \\
 & \mathbf{E}\,(D_1 \diamond (\mathbf{E}\,(D_2 \diamond E), \mathbf{E}\,(D_2 \diamond F))) \\
= & \text{“Inductive case on } D_1\text{”} \\
 & \mathbf{E}\,(D_1 \diamond \mathbf{E}\,(D_2 \diamond E)), \mathbf{E}\,(D_1 \diamond \mathbf{E}\,(D_2 \diamond F)) \\
= & \text{“Rule for sequential composition”} \\
 & \mathbf{E}\,(D_1;\; D_2 \diamond E), \mathbf{E}\,(D_1;\; D_2 \diamond F)
\end{array}
$$

For random choice assuming $0 < p < 1$ we have

$$
\begin{array}{ll}
 & \mathbf{E}\,(D_1 \;{}_p\!\boxplus\; D_2 \diamond E, F) \\
= & \text{“Rule for random choice”} \\
 & \mathbf{E}\,(D_1 \;{}_p\!\boxplus\; D_2 \diamond E), \mathbf{E}\,(D_1 \;{}_p\!\boxplus\; D_2 \diamond F)
\end{array}
$$

Here we make no appeal to the inductive case; the definition of random choice has been chosen precisely to make it work as an additional base case. □

Given Proposition 3 we can readily prove the following proposition which captures the oblivious nature of non-deterministic choice in the context of a random design.

Proposition 4 *Within a random design sequential composition distributes through non-deterministic choice, that is for any random designs D_1, D_2, D_3 and expression E*

$$\mathbf{E}\,(D_1\ ;\ D_2 \sqcap D_3 \diamond E) = \mathbf{E}\,(D_1;\ D_2\ \sqcap\ D_1;\ D_3 \diamond E)$$

where we remind the reader that choice binds more tightly than sequential composition.

Proof

$\mathbf{E}\,(D_1\ ;\ D_2 \sqcap D_3 \diamond E)$

$=$ "Rule for sequential composition"

$\mathbf{E}\,(D_1 \diamond \mathbf{E}\,(D_2 \sqcap D_3 \diamond E))$

$=$ "Rule for choice"

$\mathbf{E}\,(D_1 \diamond (\mathbf{E}\,(D_2 \diamond E), \mathbf{E}\,(D_3 \diamond E)))$

$=$ "By Proposition 3"

$\mathbf{E}\,(D_1 \diamond \mathbf{E}\,(D_2 \diamond E)), \mathbf{E}\,(D_1 \diamond \mathbf{E}\,(D_3 \diamond E))$

$=$ "Rule for sequential composition applied to each of the two terms in the bunch union"

$\mathbf{E}\,(D_1;\ D_2 \diamond E), \mathbf{E}\,(D_1;\ D_3 \diamond E)$

$=$ "Rule for choice"

$\mathbf{E}\,(D_1;\ D_2 \sqcap D_1;\ D_3 \diamond E)$ □In Proposition 4 the oblivious nature of non-deterministic choice is captured by the property that in any sequential program expressed as a random design any non-deterministic choice can be made *at the start of the program.* McIver and Morgan describe oblivious choice in exactly this way in [MM04], where, as we have noted, they obtain it in a programming logic of distributions.

We return to the question of why we obtain oblivious non-determinism by changing the definition of probabilistic choice rather than that of non-deterministic choice itself. Proposition 4, which captures the idea of oblivious choice, was proved by appeal to Proposition 3, which required a proof by structural induction over the programming connectives of random designs. Thus the oblivious nature of choice within random designs is dependent not only on the definition of choice itself, but also on the properties of the other programming connectives in the language.

8 Interactions between Feasibility and Probabilistic Choice

For non-probabilistic programs our computational model provides a backtracking interpretation. Non-deterministic choice is seen as provisional choice, with the operational interpretation that attempting to execute a command of the form $g \Longrightarrow D$ when g is false will cause execution to reverse. Execution will then continue back to a previous choice construct and then choose a previously

untried forward execution path. We use a similar approach in the case of probabilistic choice; any probabilistic choice that leads to infeasibility will be revised on reverse execution. We can exploit this behaviour in random search algorithms. Such algorithms are particularly effective where solutions are numerous but clustered together so that an exhaustive and unrandomised search might traverse most of the search space before encountering any of them. Random search algorithms will sample different parts of the state space, rather than work their way through it according to some unspecified pattern of search that might be particularly unhelpful.

We will use probabilistic choice from a set, written as $x :\oplus A$, in which each element of the set A that provides a feasible continuation is equally likely to be assigned to x. This is made more precise in the following definition, which has the restriction of applying only to finite sets. We define the effect of $x :\oplus A$ in terms of its expectation:

Definition 13

$$\begin{array}{l}\mathbf{E}\,(x :\oplus A \diamond E) = \\ \quad \textbf{let}\ \ A' = \{a \mid a \in A \wedge x := a \diamond E \neq \textbf{null}\}\ \ \textbf{and}\ \ n = \textbf{card}(A')\ \ \textbf{in} \\ \qquad n > 0 \longrightarrow \mathbf{E}\,(x := \textbf{choice}\,(A') \ {}_{\frac{1}{n}}\!\oplus\, (x :\oplus A' \setminus \{\textbf{choice}\,(A')\}) \diamond E) \\ \quad \textbf{end}\end{array}$$

In the following proposition we consider the case where E takes the form $G \longrightarrow [P]$. Such an expression is **null** where G is false, and where P is true takes the value 1 or 0 depending on whether P is true or false.

Proposition 5 *Given a finite set A and propositions G and P, define the following subsets of A:*

$$A_1 = \{a \mid a \in A \wedge (x := a \diamond G \longrightarrow [P]) = 1\}$$

$$A_2 = \{a \mid a \in A \wedge (x := a \diamond G \longrightarrow [P]) = 0\}$$

Then

$$\mathbf{E}\,(x :\oplus A \diamond G \longrightarrow [P]) = \textbf{card}(A_1 \cup A_2) > 0 \longrightarrow \frac{\textbf{card}(A_1)}{\textbf{card}(A_1)+\textbf{card}(A_2)}$$

Proof A' and n in the definition of probabilistic choice from a set are here equal to $A_1 \cup A_2$ and $\textbf{card}(A_1 \cup A_2)$ respectively. For $n = 0$ we have the LHS of the proposition is **null** from the definition of $\mathbf{E}\,(x :\oplus A \diamond E)$ and property of guarded bunches that $\textbf{false} \longrightarrow E = \textbf{null}$. The RHS of the definition is $\textbf{false} \longrightarrow 0/0$ which is also **null**.

The remaining proof is by induction on the value n. For the base case $n = 1$ we have from the definition of $\mathbf{E}\,(x :\oplus A \diamond E)$ that

$$\mathbf{E}\,(x :\oplus A \diamond G \longrightarrow [P]) \ = \ x := \textbf{choice}\,(A) \diamond G \longrightarrow [P]$$

and the goal of our proof becomes

$$x := \textbf{choice}\,(A) \diamond G \longrightarrow [P] \ = \ \frac{\textbf{card}(A_1)}{\textbf{card}(A_1)+\textbf{card}(A_2)}$$

We can relate A' in the definition of $\mathbf{E}\,(x :\oplus A \diamond E)$ to A_1 and A_2 of the proposition by $A' = A_1 \cup A_2$. For $n = 1$, A' contains a single element. We consider

cases. For $\mathbf{choice}\,(A') \in A_1$ we have both LHS and RHS of our conjecture are 1. Otherwise we must have $\mathbf{choice}\,(A') \in A_2$, and both LHS and RHS of the conjecture are 0.

For $n > 1$

$\mathbf{E}\,(x :\oplus A \diamond G \longrightarrow [P])$

$=$ "Defn of $:\oplus$"

$\mathbf{E}\,(x := \mathbf{choice}\,(A')\ {}_{\frac{1}{n}}\oplus (x :\oplus A' \setminus \{\mathbf{choice}\,(A')\}) \diamond G \longrightarrow [P])$

$=$ "Defn of ${}_p\oplus$"

$\mathbf{E}\,(x := \mathbf{choice}\,(A') \diamond G \longrightarrow [P])\ {}_p+\ \mathbf{E}\,(x :\oplus A' \setminus \{\mathbf{choice}\,(A')\} \diamond G \longrightarrow [P])$

$=$ "Since $\mathbf{choice}\,(A') \in A_1 \vee \mathbf{choice}\,(A') \in A_2$ and by inductive case"

$\mathbf{choice}\,(A') \in A_1 \longrightarrow \frac{1}{n} + \frac{n-1}{n} * \frac{\mathbf{card}(A_1)-1}{\mathbf{card}(A_1)-1+\mathbf{card}(A_2)}$,

$\mathbf{choice}\,(A') \in A_2 \longrightarrow \frac{n-1}{n} * \frac{\mathbf{card}(A_1)}{\mathbf{card}(A_1)+\mathbf{card}(A_2)-1}$

$=$ "By simplification of fractions and using rule $g \longrightarrow a, h \longrightarrow a \ = \ g \vee h \longrightarrow a$"

$\mathbf{choice}\,(A') \in A_1 \vee \mathbf{choice}\,(A') \in A_2 \longrightarrow \frac{\mathbf{card}(A_1)}{\mathbf{card}(A_1)+\mathbf{card}(A_2)}$

$=$ "Since $\mathbf{choice}\,(A') \in A_1 \vee \mathbf{choice}\,(A') \in A_2$"

$\frac{\mathbf{card}(A_1)}{\mathbf{card}(A_1)+\mathbf{card}(A_2)}$ $\square$

8.1 Case Study

We consider the specification of a point search:

$$D_A \ \widehat{=}\ f \in X \nrightarrow Y \wedge y \in \mathbf{ran}\,(f) \vdash f(x') = y$$

and show that it has an implementation

$$D_C \ \widehat{=}\ x :\oplus \mathbf{dom}\,(f)\,; f(x) = y \Longrightarrow I\!I$$

We need to show $\mathbf{E}\,(D_C \diamond [f(x) = y]) = 1$ under the assumption $f \in X \nrightarrow Y \wedge y \in \mathbf{ran}\,(f)$.

LHS

$=$ "By Defn of D_C, Sequential Composition, Guard and Skip"

$\mathbf{E}\,(x :\oplus \mathbf{dom}\,(f) \diamond f(x) = y \longrightarrow [f(x) = y])$

$=$ "By Proposition 1"

$\frac{\mathbf{card}(A_1)}{\mathbf{card}(A_1)+\mathbf{card}(A_2)}$

where:

$A_1 = \{a \mid a \in \mathbf{dom}\,(f) \wedge (x := a \diamond f(x) = y \longrightarrow [f(x) = y]) = 1\}$
$A_2 = \{a \mid a \in \mathbf{dom}\,(f) \wedge (x := a \diamond f(x) = y \longrightarrow [f(x) = y]) = 0\}$

by assignment rule these sets are:

$A_1 = \{a \mid a \in \mathbf{dom}\,(f) \wedge (f(a) = y \longrightarrow [f(a) = y]) = 1\}$

$$A_2 = \{a \mid a \in \mathbf{dom}\,(f) \wedge (f(a) = y \longrightarrow [f(a) = y]) = 0\}$$

Now observing that an expression with form $P \longrightarrow [P]$ never takes the value 0, we see $\mathbf{card}(A_2) = 0$. Also since $y \in \mathbf{ran}\ (f)$ there will be some a with $a \in \mathbf{dom}\,(f)$ and $f(a) = y$ so $\mathbf{card}(A_1) > 0$, hence:

$$\frac{\mathbf{card}(A_1)}{\mathbf{card}(A_1)+\mathbf{card}(A_2)} = 1 \quad \text{and thus LHS} = 1 \text{ as required.} \quad \square$$

We can calculate the probability of obtaining a particular x' after D_c as follows:

$\mathbf{prob}_{D_C}(x = x')$

$=$ "Defn of $\mathbf{prob}_D$"

$\mathbf{E}\,(D_C \diamond [x = x'])$

$=$ "Defn of D_C, Sequential Composition, Guard and Skip"

$\mathbf{E}\,(x :\oplus \mathbf{dom}\,(f) \diamond f(x) = y \longrightarrow [x = x'])$

$=$ "By Proposition 1"

$\frac{\mathbf{card}(A_1)}{\mathbf{card}(A_1)+\mathbf{card}(A_2)}$ where

$A_1 = \{a \mid a \in \mathbf{dom}\,(f) \wedge (x := a \diamond f(x) = y \longrightarrow [x = x']) = 1\}$ and

$A_2 = \{a \mid a \in \mathbf{dom}\,(f) \wedge (x := a \diamond f(x) = y \longrightarrow [x = x']) \neq 1\}$

by substitution and bunch properties these sets are:

$$A_1 = \{a \mid a \in \mathbf{dom}\,(f) \wedge f(a) = y \wedge x = x'\}$$
$$A_2 = \{a \mid a \in \mathbf{dom}\,(f) \wedge f(a) = y \wedge x \neq x'\}$$

and hence

$A_1 \cup A_2$

$=$ "Set union property $\{x \mid P \wedge Q\} \cup \{x \mid P \wedge \neg\, Q\} = \{x \mid P\}$"

$\{a \mid a \in \mathbf{dom}\,(f) \wedge f(a) = y\}$

$=$ "Since $f^{-1}(y)$ denotes the bunch of elements f maps to y"

$\{f^{-1}(y)\}$

For $f(x') = y$ we have $\mathbf{card}(A_1) = 1$ and for $f(x') \neq y$ we have $\mathbf{card}(A_1) = 0$, and noting that since A_1 and A_2 are disjoint $\mathbf{card}(A_1)+\mathbf{card}(A_2) = \mathbf{card}(A_1 \cup A_2)$ we have

$$\mathbf{prob}_{D_C}(x = x') \;=\; \frac{\mathbf{card}(A_1)}{\mathbf{card}(A_1 \cup A_2)} \;=\; \frac{[f(x')=y]}{\mathbf{card}\,\{f^{-1}(y)\}}$$

a result that tells us it is impossible to obtain an x' for which $f(x') \neq y$ and that each value for which $f(x') = y$ is equally likely to be obtained.

9 Conclusions and Future Work

We have added probabilistic choice to our design-based theory of reversible computation. In our original theory we gave rules, defined over the syntactic constructs of our language, which give the prospective values an expression could take after an operation is executed. The rules were proved from a closed-form definition. In this paper we introduce probabilism by adding probabilistic

choice and converting our prospective-value rules to yield expectations. We do not give a full semantic justification for our rules here, but they have certain properties which provide a level of confidence in their correctness and indicate how that confidence could be further increased. First, for designs which do not contain probabilistic choice, expected values reduce to prospective values. Second, we use our rules for expectation to extract the distributions associated with a design and to express its associated relation. That in turn, is used to obtain a derivation of expectations from relations, closing a circle that could be used to validate our expectation calculus with respect to a relational model.

Refinement is defined in terms of the containment of convex closures in distribution space. However, since our approach allows the expression of possibly infeasible operations, we must qualify our refinement rule with an additional predicate that prevents over-refinement, i.e. refinement to the point of infeasibility.

We are able to link our new probabilistic designs to non-probabilistic designs by a Galois connection. To transform from the richer theory of probabilistic designs to standard designs we treat probabilistic choice as non-deterministic choice. The reverse transform lifts values into point distributions. The existence of the Galois connection means we can apply standard reasoning to probabilistic designs by treating probabilistic choice as non-deterministic choice.

In our formulation we find the same, initially counter-intuitive, interaction between probabilistic and demonic choice as reported by other workers. On reflection, this interaction is necessary and derives from the demon's ability to make a choice that depends on the current state. Sometimes, however, an oblivious form of non-determinism is required, and we give an alternative formulation which achieves this.

An important aspect of our approach is that probabilistic choice is governed by feasibility. Operationally, a probabilistic choice may indeed select a continuation which is not feasible, but this will result in execution reversing back to the point of choice, and another alternative being selected if one is available. A short case study is provided to show how this interaction between feasibility and probabilistic choice may be exploited in terms of practical programming.

One aspect omitted from this paper, apart from one brief remark, is the alphabetisation of probabilistic relations. This cannot be expressed solely in terms of predicates on before states and after states, since that can only describe homogeneous relations. We therefore need to introduce a new after-state variable ranging over distributions. The elaboration of these predicates and the rules for their combination over the syntactic constructs of our language remains as future work. Other "unfinished business" includes a formal exploration of the monotonicity properties of probabilistic star-refinement and an exploration of a link between our language with oblivious non-determinism and other languages which exhibit the same property.

Acknowledgments

We thank the referees of the published version of this report for their careful and patient reading of our original text and for their many invaluable comments.

References

[AM02] J.-R. Abrial and W. J. Mussat. On Using Conditional Definitions in Formal Theories. In *ZB 2002: Formal Specification and Development in Z and B*, volume 2272 of *Lecture Notes in Computer Science*. Springer, 2002.

[Ben73] C Bennett. The Logical Reversibility of Computation. *IBM Journal of Research and Development*, 6, 1973.

[Ben82] C Bennett. The Thermodynamics of Computation. *International Journal of Theoretical Physics*, 21 pp 905-940, 1982.

[dH01] J. den Hartog. *Probabilistic Extensions of Semantic Models*. PhD thesis, IPA, Amsterdam, 2001.

[Fey96] R P Feynman. *Lectures on Computation.* Westview Press, 1996.

[Flo67] R. W. Floyd. Nondeterministic Algorithms. *Journal of the ACM*, 14(4):636–644, October 1967.

[Heh81] E. C. R. Hehner. Bunch Theory: a Simple Set Theory for Computer Science. *Information Processing Letters*, 12(1):26–30, February 1981.

[Heh93] E. C. R. Hehner. *A practical theory of programming*. Texts and Monographs in Computer Science. Springer, 1993.

[HS06] J. He and J. W. Sanders. Unifying Probability. In *First International Symposium on Unifying Theories of Programming, UTP 2006*, volume 4010 of *Lecture Notes in Computer Science*, pages 173–199. Springer, 2006.

[HSM97] J. He, K. Seidel, and A. McIver. Probabilistic Models for the Guarded Command Language. *Science of Computer Programming*, 28(2-3):171–192, 1997.

[Lan61] R Landauer. Irreversibility and Heat Generated in the Computing Process. *IBM J R&D*, 5, 1961.

[MM04] A. McIver and C. C. Morgan. *Abstraction, Refinement and Proof For Probabilistic Systems*. Springer, 2004.

[Mor88] C. C. Morgan. The Specification Statement. *ACM Transactions on Programming Languages and Systems*, 10(3):403–419, July 1988.

[Nel89] G. Nelson. A Generalization of Dijkstra's Calculus. *ACM Transactions on Programming Languages and Systems*, 11(4):517–561, October 1989.

[SDG99] W. J. Stoddart, S. E. Dunne, and A. J. Galloway. Undefined Expressions and Logic in Z and B. *Formal Methods in System Design*, 15(3), 1999.

[Sto06] W. J. Stoddart. The Reversible Virtual Machine. User and technical manuals, University of Teesside, UK, July 2006.

[SZL06] W. J. Stoddart, F. Zeyda, and A. R. Lynas. A Design-Based Model of Reversible Computation. In *UTP 2006, First International Symposium on Unifying Theories of Programming*, volume 4010 of *Lecture Notes in Computer Science*, pages 63–83. Springer, June 2006.

[VWar] D. Varacca and G. Winskel. Distributing probability over nondeterminism. *Mathematical Structures in Computer Science*, to appear.

[Zey07] F. Zeyda. *Reversible Computation in B.* PhD thesis, University of Teesside, 2007.

[ZSD03] F. Zeyda, W. J. Stoddart, and S. Dunne. The Refinement of Reversible Computations. In *RCS'03: 2nd International Workshop on Refinement of Critical Systems: Methods, Tools and Developments*, January 2003.

[ZSD05] F. Zeyda, W. J. Stoddart, and S. Dunne. A Prospective-Value Semantics for the GSL. In *ZB 2005: Formal Specification and Development in Z and B*, volume 3455 of *Lecture Notes in Computer Science*, pages 187–202. Springer, April 2005.

[Zul01] P Zuliani. Logical reversibility. *IBM J R&D*, 45(6), 2001.