



The United Nations
University

UNU-IIST

International Institute for
Software Technology

Refinement Algebra with Explicit Probabilism

T. M. Rabeaja and J. W. Sanders

February 2009

UNU-IIST and UNU-IIST Reports

UNU-IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macao, and was founded in 1991. It started operations in July 1992. UNU-IIST is jointly funded by the government of Macao and the governments of the People's Republic of China and Portugal through a contribution to the UNU Endowment Fund. As well as providing two-thirds of the endowment fund, the Macao authorities also supply UNU-IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU-IIST is to assist developing countries in the application and development of software technology.

UNU-IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU-IIST, and
7. Dissemination, in which UNU-IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU-IIST is on formal methods for software development. UNU-IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU-IIST produces a report series. Reports are either Research [R], Technical [T], Compendia [C] or Administrative [A]. They are records of UNU-IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU-IIST at P.O. Box 3058, Macao or visit UNU-IIST's home page: <http://www.iist.unu.edu>, if you would like to know more about UNU-IIST and its report series.

Govindan Parayil, Director, a.i.



The United Nations
University

UNU-IIST

International Institute for
Software Technology

P.O. Box 3058
Macao

Refinement Algebra with Explicit Probabilism

T. M. Rabeaja and J. W. Sanders

Abstract

Refinement algebra provides axioms for the stepwise removal of abstraction, in the form of demonic nondeterminism, in a first-order system that supports reasoning about loops. It has been extended by Solin and Meinicke to computations involving implicit probabilistic choices: demonic nondeterminism then satisfies weaker properties. In this paper their axiom system is extended to capture explicit probabilistic choices. The first form is an unquantified probabilistic choice; the second is a partial quantified probabilistic choice (from which the usual binary probabilistic choice can be recovered). The new refinement algebra is sound with respect to 1-bounded expectation transformers, the premier model of probabilistic computations, but also with respect to a new model introduced here to capture more directly partial quantified computations. In this setting a ‘normal form’ result of Kozen is proved, replacing multiple loops with a single loop that does the same job; and the extent to which the two forms of loop have the same expected number of steps to termination is considered. Being entirely within first-order logic, the new refinement algebra is targetted to automation.

Tahiry Rabehaja is a UNU-IIST Fellow. He graduated from the University of Antananarivo (Madagascar) with a B.Sc (Mathematics) degree in August 2006, and obtained a Post-Graduate diploma in Mathematical Sciences from the African Institute for Mathematical Sciences in June, 2008.

Jeff Sanders is Principal Research Fellow at UNU-IIST. His interests lie largely in Formal Methods.

The authors acknowledge assistance from the Macao Science and Technology Development Fund under the PEARL project, grant number 041/2007/A3.

Contents

1	Introduction	1
2	Axiomatization	1
2.1	Probabilistic demonic refinement algebra	1
2.2	Probabilistic refinement algebra	2
3	Models and soundness	4
3.1	A transformer model	4
3.2	Distributional models	4
3.3	New models	5
4	Probabilistic normal-form theorem	6
4.1	Guards and assertions	7
4.2	Transformation 1: conditional	8
4.3	Transformation 2: nested loop	10
4.4	Transformation 3: removing post-computation	10
4.5	Transformation 4: composition	11
4.6	Transformation 5: probability	12
5	Example: iterated random jumps	13
6	Conclusion	15

1 Introduction

The primary motivation for an algebraic approach to programming is its simplicity. It is possible, surprisingly, to find a small number of simple properties (first order, and hence automatable—see for example [4]) that are powerful enough to facilitate the difficult task of reasoning about loops.

Anticipated by Conway [2] (under the name ‘regular algebra’) to tackle Kleene’s quest for a complete axiomatization for regular expressions, the axioms for Kleene algebra were adopted and studied by Kozen who showed [6] that they solve Kleene’s problem and used them [7] to give an algebraic proof that a program with multiple loops can be transformed to a functionally equivalent one with a single loop. Kleene algebra was extended by Cohen [1] to express infinite iteration, and presented by von Wright [13] in the form, exploited here, of a refinement algebra involving ‘strong iteration’ to facilitate total correctness reasoning.

Reflecting recent interest in reasoning about probabilistic programs, a variant of Kleene algebra has been used by McIver, Cohen and Morgan [8]. By contrast Solin [11] and Meinicke [10] have extended von Wright’s refinement algebra to reason about nondeterminism in the presence of implicit probabilism. Here that setting is further extended by the introduction of two operators incorporating probability explicitly: an unquantified and a partially quantified probabilistic choice. Models based on the expectation transformer and distributional models are provided to ensure the soundness of the axiomatization.

It is shown that the resulting refinement algebra is strong enough to permit an algebraic proof of a probabilistic version of a folk theorem for multiple loops, within a total correctness setting [11]. By semantic reasoning, the two programs have the same expected number of steps to termination. Finally, an example of a probabilistic program modelling random jumps is presented. Its normal form is given and shown, by analytic reasoning, to have the same expected number of step to termination.

2 Axiomatization

2.1 Probabilistic demonic refinement algebra

Following Solin [11] and Meinicke [10], a *probabilistic refinement algebra* with carrier set X is defined to be a structure with signature $(X, \sqcap, \wp, *, \omega, \top, \mathbb{I})$ which satisfies the axioms in Figure 1. The binary operator \sqcap represents nondeterministic choice. Sequential composition $x \wp y$ is usually abbreviated xy . The unary operators $*$ and ω for iteration respectively represent ‘weak’ (*i.e.* finite, so capturing partial correctness) and ‘strong’ (*i.e.* countable, so capturing total correctness) iteration. The constant \top represents the greatest computation **magic** (not a program) and \mathbb{I} stands for the ineffectual program **skip**.

$$\begin{aligned}
x \sqcap (y \sqcap z) &= (x \sqcap y) \sqcap z & (1) \\
x \sqcap y &= y \sqcap x & (2) \\
x \sqcap x &= x & (3) \\
x \sqcap \top &= x & (4) \\
x(yz) &= (xy)z & (5) \\
\mathbb{I}x &= x = x\mathbb{I} & (6) \\
\top x &= x & (7) \\
x(y \sqcap z) &\sqsubseteq xy \sqcap xz & (8) \\
(x \sqcap y)z &= xz \sqcap yz & (9) \\
x^* &= \mathbb{I} \sqcap xx^* & (10) \\
x \sqsubseteq yx \sqcap z &\implies x \sqsubseteq y^*z & (11) \\
x \sqsubseteq x(y \sqcap \mathbb{I}) \sqcap z &\implies x \sqsubseteq zy^* & (12) \\
x^\omega &= \mathbb{I} \sqcap xx^\omega & (13) \\
yx \sqcap z \sqsubseteq x &\implies y^\omega z \sqsubseteq x & (14)
\end{aligned}$$

Figure 1: The axioms for a *probabilistic demonic refinement algebra*. Variables x, y, z range over the carrier set and $x \sqsubseteq y$ iff $x \sqcap y = x$.

The divergent program **abort** is defined $\perp := \mathbb{I}^\omega$ and a ready calculation shows that for any x in the carrier set

$$\perp x = \perp \quad \text{and} \quad \perp \sqsubseteq x.$$

All of the operators are monotonic (from both the left and right in the binary case). For instance, sequential composition is monotonic by Laws (8) and (9), and monotonicity of ω follows from

$$\begin{aligned}
x \sqsubseteq y & \\
\implies & \hspace{15em} \text{monotonicity of } \mathbb{I} \text{ and } \sqcap \\
\mathbb{I} \sqcap xy^\omega \sqsubseteq \mathbb{I} \sqcap yy^\omega & \\
\implies & \hspace{15em} \text{Axioms (13) and (14)} \\
x^\omega \sqsubseteq y^\omega. &
\end{aligned}$$

The properties of sequential composition, conditional and tail recursion are fully modelled within that algebra [13] but explicit probabilistic choices are not (like a random choice within a finite set or like a probabilistic branch). One way of solving that problem is to introduce the operators \mathbb{P}^p , where p is a $[0, 1]$ -valued expression (of state), to represent a binary choice with probability p .

2.2 Probabilistic refinement algebra

The approach taken here, to the inclusion of explicit probabilism in a refinement algebra, is to introduce two operators. That choice has been guided by the possibility of implementing the

$$\begin{aligned}
x \oplus (y \oplus z) &= (x \oplus y) \oplus z & (15) \\
x \oplus y &= y \oplus x & (16) \\
x \oplus (y \sqcap z) &= (x \oplus y) \sqcap (x \oplus z) & (17) \\
(y \oplus z)x &= yx \oplus zx & (18) \\
p(qx) &= (pq)x & (19) \\
p(xy) &= (px)y & (20) \\
1x &= x & (21) \\
(p + q)x &= px \oplus qx & (22)
\end{aligned}$$

Figure 2: Further axioms required for a *probabilistic refinement algebra*. Again x, y, z range over the carrier set and p, q over Λ .

axioms with only unary and binary operators in an automatic theorem prover, as in [4]. The first operator, \oplus , represents an unquantified probabilistic choice: made with some undisclosed probability. The second, \cdot , represents a partial, quantified, probabilistic choice: it forms a *commutative semi-ring* $(\Lambda, +, \cdot, 0, 1)$ where Λ is the set of (state-dependent) probabilities $\Lambda := S \rightarrow [0, 1]$ and $+$ is defined pointwise on Λ and allowed to be partially defined, though any $p \in \Lambda$ has some $\bar{p} \in \Lambda$ satisfying $p + \bar{p} = 1$. The outer product $p \cdot x$ is abbreviated px , where by convention $p, q, \dots : \Lambda$ and x, y, \dots are elements of the carrier set. The operator \oplus is partially defined but $px \oplus qy$ is always defined when $p + q$ is.

A *probabilistic refinement algebra over Λ* is a structure with signature $(X, \sqcap, \wp, \oplus, \cdot, *, \omega, \top, \mathbb{I})$ for which the restriction to $(X, \sqcap, \wp, *, \omega, \top, \mathbb{I})$ is a probabilistic demonic refinement algebra and furthermore the axioms of Figure 2 hold.

Law (17) ensures that \oplus is monotonic. The refinement $p(x \sqcap y) \sqsubseteq px \sqcap py$ follows from (6), (8) and (20); so \cdot is also monotonic in its right-hand argument. Moreover, idempotence $px \oplus \bar{p}x = x$ follows from (21) and (22) and the usual binary probabilistic choice is recovered $x \wp y := px \oplus \bar{p}y$.

Other properties such as

$$x(py \oplus \bar{p}z) \sqsupseteq pxy \oplus \bar{p}xz$$

could be added if needed but axioms (1) – (22) suffice here.

A simple consequence of these axioms is the following refinement (and its extension to any finite number of terms).

Lemma 2.1. *For any x, y in the carrier set and $p : \Lambda$, $x \sqcap y \sqsubseteq px \oplus \bar{p}y$.*

Proof. By definition of \sqcap , $x \sqcap y \sqsubseteq x$ and $x \sqcap y \sqsubseteq y$. Since \cdot is monotonic, $p(x \sqcap y) \sqsubseteq px$ and $\bar{p}(x \sqcap y) \sqsubseteq \bar{p}y$. So by idempotence and monotonicity of \oplus ,

$$x \sqcap y = p(x \sqcap y) \oplus \bar{p}(x \sqcap y) \sqsubseteq px \oplus \bar{p}y.$$

□

3 Models and soundness

3.1 A transformer model

The simplest nontrivial model of a probabilistic refinement algebra is the set of *uniformly* bounded expectation transformers over state space S with $\Lambda := S \rightarrow [0, 1]$. Normalizing, let the set of *1-bounded (positive) expectations* be $\mathbb{E}_1 := S \rightarrow [0, 1]$. A *1-bounded expectation transformer* [9] is an element of $\mathbb{T}_1 := \mathbb{E}_1 \rightarrow \mathbb{E}_1$. For any $c : [0, 1]$, the constant expectation of value c is denoted \mathbf{c} . The pointwise ordering on \mathbb{E}_1 induces a pointwise order in \mathbb{T}_1 which is the interpretation of refinement. The constants are interpreted: \top as $\lambda e : \mathbb{E}_1 \cdot \mathbf{1}$; \perp as $\lambda e : \mathbb{E}_1 \cdot \mathbf{0}$; and \mathbb{I} as $id_{\mathbb{E}_1}$.

The binary operators \circledast , \sqcap and \oplus are interpreted, respectively, as functional composition, minimum and functional sum. For $t \in \mathbb{T}_1$, $p : S \rightarrow [0, 1]$ and $e \in \mathbb{E}_1$, the partial probabilistic choice \cdot is

$$(pt).e := \lambda s : S \cdot (p.s)(t.e.s).$$

In this section, function application $f(x)$ appears as $f.x$. The unary operators $*$ and ω have their usual meanings as respectively the greatest and least fixed points of the function $F_t : \mathbb{T}_1 \rightarrow \mathbb{T}_1$ defined

$$F_t.u := \mathbb{I} \sqcap t \circ u.$$

3.2 Distributional models

The next two sections present a way of constructing a model of (1) – (22) from a structure containing the operators \circledast , \sqcap and \oplus which is inspired by Varacca and Winskel [12].

First recall the definition of the distributional model for probabilistic computation [5]. Given a (finite) set of states S , the set of subdistributions over S is defined

$$\bar{S} := \{x : S \rightarrow [0, 1] \mid \sum x.s \leq 1\}$$

and ordered pointwise. Denoting by $C\bar{S}$ the set of subsets of \bar{S} which are convex and up-closed, the distributional model of probabilistic computations is defined $\mathbb{H} := S \rightarrow C\bar{S}$. It is ordered by set containment, *i.e.*

$$r \sqsubseteq r' := \forall s : S \cdot r.s \supseteq r'.s.$$

For $r, r' \in \mathbb{H}$, one defines [5]:

$$\begin{aligned} (r \circledast r').s &:= cc \uparrow \cup \{\sum (r'.s)y.s \mid y \in r.s\} \\ (r_p \oplus r').s &:= (p.s)(r.s) + (\bar{p}.s)(r'.s) \\ (r \sqcap r').s &:= cc(r.s \cup r'.s) \end{aligned}$$

where cc and \uparrow denote, respectively, convex and up-closure. The order of applying closures does not matter [5] since $cc(\uparrow A) = \uparrow(ccA)$.

The constants are interpreted: \top as $\lambda s : S \cdot \emptyset$; \perp as $\lambda s : S \cdot \overline{S}$; and \mathbb{I} as $\lambda s : S \cdot \{\delta_s\}$, using the conventions concerning functions:

$$0\emptyset := \{\mathbf{0}\}, \quad \forall p > 0 \cdot p\emptyset := \emptyset \quad \text{and} \quad \emptyset + A := \emptyset.$$

Alternatively, the construction of Section 3.1 can be generalized by considering expectations $\mathbb{E} := S \rightarrow [0, +\infty)$ (or $[0, +\infty]$) whence the model becomes $\mathbb{T} := \mathbb{E} \rightarrow \mathbb{E}$ and involves the operators \circ, \sqcap and \oplus .

3.3 New models

The purpose of this section is to construct new models of probabilistic refinement algebra that more closely model partial probabilistic computations.

Let \mathcal{M} be a model containing the operators \oplus . One considers $\Lambda : S \rightarrow [0, 1]$ and the *projection* $\pi : \mathcal{M} \times \Lambda \rightarrow \mathcal{M}$ defined

$$\pi(x, p) := x_p \oplus \perp.$$

Thus the couple (x, p) represents execution of x with probability p and abortion with probability \bar{p} (Hence the term *partial quantified computation*). Now, the operators of \mathcal{M} in $\mathcal{M} \times \Lambda$ can be extended

$$\begin{aligned} (x, p) \circ (y, q) &:= (x \circ \pi(y, q), p) \\ (x, p) \sqcap (y, q) &:= \left(\pi \left(x, \frac{p}{p \sqcup q} \right) \sqcap \pi \left(y, \frac{q}{p \sqcup q} \right), p \sqcup q \right) \\ (x, p) \oplus (y, q) &:= \left(x_{\frac{p}{p+q}} \oplus y, p + q \right). \end{aligned}$$

Notice that in the last definition, \oplus is well defined only if $p + q \leq 1$; but if undefined it can sometimes be extended to be fully defined, as a result of the following construction.

Together with π , one considers $\varepsilon : \mathcal{M} \rightarrow \mathcal{M} \times \Lambda$ such that $\varepsilon.x := (x, \mathbf{1})$. A straightforward calculation shows that (ε, π) is a *Galois embedding*. Constants of $\mathcal{M} \times \Lambda$ are the respective images of the constants of \mathcal{M} and unary operators are defined

$$(x, p)^\bullet := \varepsilon(\pi(x, p)^\bullet) \quad \text{for} \quad \bullet \in \{*, \omega\}.$$

From those definitions, $(\varepsilon.\mathbb{I})^\omega = (\mathbb{I}^\omega, \mathbf{1}) = \varepsilon.\perp$. Finally, product is defined $p(x, q) := (x, pq)$.

Notice there is a problem of division by 0 in the definition of \sqcap and \oplus . Firstly, $(x, \mathbf{0}) \oplus (y, p) = (y, p)$ for any $x, y \in \mathcal{M}$ and $p > \mathbf{0}$. Then one assumes that $(x, \mathbf{0})$ is *equivalent* to $(y, \mathbf{0})$ for any x, y . More generally, consider the equivalence relation

$$\begin{aligned} &(x, p) \sim (y, q) \\ &\text{iff} \\ &\forall u : \Lambda \cdot (p \sqcap q \leq u \wedge u > \mathbf{0}) \Rightarrow \pi(x, p/u) = \pi(y, q/u). \end{aligned}$$

Indeed, that equivalence is a *congruence*. Moreover $(x, p) \sim (y, q) \Rightarrow \pi(x, p) = \pi(y, q)$ and equivalence holds if $((x_p \oplus \perp = y_p \oplus \perp \wedge p > \mathbf{0}) \Rightarrow x = y)$. Secondly, by convention $(x, \mathbf{0}) \sqcap (y, \mathbf{0}) = (\perp, \mathbf{0})$ and the definition of \oplus is extended accordingly to \sim .

The semantic space is defined to be the quotient $\overline{\mathcal{M}} := (\mathcal{M} \times \Lambda) / \sim$.

Theorem 3.1. $(\overline{\mathcal{M}}, \sqcap, \wp, \oplus, \cdot, *, \omega, \varepsilon, \top, \varepsilon, \mathbb{I})$ is a probabilistic refinement algebra over $(\Lambda, +, \cdot, \mathbf{0}, \mathbf{1})$.

Proof. Proofs of the properties of the unary and binary operators are a matter of straightforward calculation (requiring 0 as a special case, frustratingly) in the product $\mathcal{M} \times \Lambda$, which implies the corresponding properties in $\overline{\mathcal{M}}$. For the constant ε, \mathbb{I} ,

$$(\mathbb{I}, \mathbf{1}) \wp (x, p) = (x_p \oplus \perp, \mathbf{1}) \sim (x, p) = (x, p) \wp (\mathbb{I}, \mathbf{1}).$$

The reasoning for ε, \top is similar. □

In [12], Varacca and Winskel showed there is no idempotent and distributive operator \oplus (*i.e.* satisfying (17) and (22)) over a power set construction. In their proof non-deterministic choice is represented by set union but in the model \mathbb{H} , it is modelled by the convex closure of the union.

These models ensure that our axiomatization is sound.

4 Probabilistic normal-form theorem

The purpose of this section is to show algebraically that any probabilistic program with multiple loops can be transformed to a functionally equivalent program with a single loop. That result, in the probability-free setting, is used as an example of folklore by Harel to whose entertaining summary [3] one refers for the history, which includes its place in ‘the structure theorem’ of structured programming and attempted proofs using transformation to ‘normal form’. In the probability-free setting it was proved algebraically by Kozen [7] using ‘Kleene Algebra with tests’ to establish partial correctness. That result was extended to total correctness by Solin [11]. Here the transformational approach inherited by Kozen and Solin is extended to cover totally correct probabilistic programs.

4.1 Guards and assertions

First, the algebraic versions of guard and assertion are needed. An element b of the carrier set is a guard [11] if it has a *conjunctive complement* \bar{b} :

$$\begin{aligned}\bar{b}(x \sqcap y) &= \bar{b}x \sqcap \bar{b}y \\ b \sqcap \bar{b} &= \mathbb{I} \\ b\bar{b} &= \top.\end{aligned}$$

Since guards are then conjunctive, they form a Boolean algebra with respect to the structure $(\sqcap, \bar{\cdot}, \top, \mathbb{I}, \neg)$. To each guard is associated the assertion

$$b^\circ := \bar{b}\perp \sqcap \mathbb{I}.$$

In dealing with total correctness, Kozen's 'strong' commutativity condition $xb = bx$ is no longer satisfiable (when x does not terminate) but the refinement $xb \sqsubseteq bx$ still holds. That, then, replaces the strong commutativity condition. An element x of the carrier set is said to *preserve* b iff $xb \sqsubseteq bx$ and $x\bar{b} \sqsubseteq \bar{b}x$ (see [11]).

Firstly, one needs the following results of Solin [11].

Lemma 4.1. *If x, y, z are elements of the carrier set then*

- (i) $(x \sqcap y)^\omega = x^\omega (yx^\omega)^\omega$
- (ii) $x(yx)^\omega \sqsubseteq (xy)^\omega x$ with equality if x is conjunctive
- (iii) if $b^\circ yx \sqsubseteq xc^\circ z$ and $x\bar{c} \sqsubseteq \bar{b}x$, then $(by)^\omega \bar{b}x \sqsubseteq x(cz)^\omega \bar{c}$.

Then

Lemma 4.2. *If b is a guard and x an element of the carrier set then*

- (i) $xb \sqsubseteq bx$ iff $bx b = bx$
- (ii) if x preserves b then $(bx)^\omega b = bx^\omega$
- (iii) if x preserves b then $b(bx)^\omega \sqsubseteq b(bx \sqcap \bar{b}y)^\omega$.

Proof. (i) The reverse implication follows from right distributivity and the direct one from left subdistributivity and monotonicity of $\bar{\cdot}$.

(ii) Since b is conjunctive, \sqsubseteq follows by [11] page 107. The reverse refinement follows from

$$x \sqsubseteq xb$$

$$\Rightarrow$$

 monotony of ω and \wp

$$bx^\omega \sqsubseteq b(xb)^\omega$$

$$\Rightarrow$$

 Lemma 4.1.(ii), b conjunctive

$$bx^\omega \sqsubseteq (bx)^\omega b$$

(iii) Since guards are conjunctive and since the proof in [11] page 111 uses only right distributivity, the result follows. \square

Sometimes, the value of some test b must be preserved across some computation x . Indeed later it will be necessary to introduce the program $s(bc \sqcap \bar{b}\bar{c})$ where c is a new guard preserved by x . The program s can be thought of as a computation which assigns the value of b to some fresh variable which is then tested in c .

For convenience, the conditional **if** $b \rightarrow x \sqcap \bar{b} \rightarrow \mathbf{skip}$ **fi** is written **if** $b \rightarrow x$ **fi**.

At last one may consider the program transformations required to convert a program to *normal form* having a single loop. Of the five used here, the first four are Kozen-like [7].

4.2 Transformation 1: conditional

If c is preserved by x_1, x_2, y_1 and y_2 then

$$\begin{aligned} & s \wp bc \sqcap \bar{b}\bar{c} \wp \\ & \mathbf{if} \ b \rightarrow x_1 \wp \ \mathbf{do} \ b_1 \rightarrow y_1 \ \mathbf{od} \\ & \sqcap \ \bar{b} \rightarrow x_2 \wp \ \mathbf{do} \ b_2 \rightarrow y_2 \ \mathbf{od} \\ & \mathbf{fi} \\ & = \\ & s \wp bc \sqcap \bar{b}\bar{c} \wp \\ & \mathbf{if} \ c \rightarrow x_1 \\ & \sqcap \ \bar{c} \rightarrow x_2 \\ & \mathbf{fi} \wp \\ & \mathbf{do} \ cb_1 \sqcap \bar{c}b_2 \rightarrow \ \mathbf{if} \ c \rightarrow y_1 \\ & \qquad \sqcap \ \bar{c} \rightarrow y_2 \\ & \qquad \mathbf{fi} \\ & \mathbf{od} . \end{aligned}$$

One can assume s as part of the precomputation since these programs are equivalent if those without s are. In the algebra, one has to show that the last term of

$$\begin{aligned}
 & (bc \sqcap \bar{b}\bar{c}) [bx_1(b_1y_1)^\omega \bar{b}_1 \sqcap \bar{b}x_2(b_2y_2)^\omega \bar{b}_2] \\
 = & \hspace{10em} bc \sqcap \bar{b}\bar{c} \text{ is conjunctive, guards form a Boolean algebra} \\
 & bcx_1(b_1y_1)^\omega \bar{b}_1 \sqcap \bar{b}\bar{c}x_2(b_2y_2)^\omega \bar{b}_2
 \end{aligned}$$

is equal to the last term of

$$\begin{aligned}
 & (bc \sqcap \bar{b}\bar{c})(cx_1 \sqcap \bar{c}x_2)[(cb_1 \sqcap \bar{c}b_2)(cy_1 \sqcap \bar{c}y_2)]^\omega (\overline{cb_1 \sqcap \bar{c}b_2}) \\
 = & \hspace{10em} \text{guards are conjunctive, } \overline{cb_1 \sqcap \bar{c}b_2} = \bar{c}\bar{b}_1 \sqcap \bar{c}\bar{b}_2 \\
 & (bcx_1 \sqcap \bar{b}\bar{c}x_2)(cb_1y_1 \sqcap \bar{c}b_2y_2)^\omega (\bar{c}\bar{b}_1 \sqcap \bar{c}\bar{b}_2) \\
 = & \hspace{10em} \text{right distributivity (9), } B = \bar{c}\bar{b}_1 \sqcap \bar{c}\bar{b}_2 \\
 & bcx_1(cb_1y_1 \sqcap \bar{c}b_2y_2)^\omega B \sqcap \bar{b}\bar{c}x_2(cb_1y_1 \sqcap \bar{c}b_2y_2)^\omega B.
 \end{aligned}$$

For the direct refinement,

$$\begin{aligned}
 & bcx_1(cb_1y_1 \sqcap \bar{c}b_2y_2)^\omega B \\
 \sqsupseteq & \hspace{10em} cx_1 = cx_1c, \text{ Lemma 4.2(iii)} \\
 & bcx_1c(cb_1y_1)^\omega B \\
 = & \hspace{10em} \text{Lemma 4.2(ii), } cc = c, B = \bar{c}\bar{b}_1 \sqcap \bar{c}\bar{b}_2 \\
 & bcx_1(cb_1y_1)^\omega c(\bar{c}\bar{b}_1 \sqcap \bar{c}\bar{b}_2) \\
 = & \hspace{10em} c \text{ is conjunctive, guards form a Boolean algebra} \\
 & bcx_1(cb_1y_1)^\omega \bar{c}\bar{b}_1 \\
 = & \hspace{10em} c \text{ is preserved by } x_1, y_1, \text{ Lemma 4.2(ii)} \\
 & bcx_1(b_1y_1)^\omega \bar{b}_1.
 \end{aligned}$$

Similarly,

$$\bar{b}\bar{c}x_2(b_2y_2)^\omega \bar{b}_2 \sqsubseteq \bar{b}\bar{c}x_2(cb_1y_1 \sqcap \bar{c}b_2y_2)^\omega B$$

and the result follows from monotonicity of \sqcap .

For the reverse refinement,

$$\begin{aligned}
 & cb_1y_1 \sqcap \bar{c}b_2y_2 \sqsubseteq cb_1y_1 \\
 \Rightarrow & \hspace{10em} \text{monotonicity of } \omega \text{ and } \sqcap \\
 & bcx_1(cb_1y_1 \sqcap \bar{c}b_2y_2)^\omega B \sqsubseteq bcx_1(cb_1y_1)^\omega B \\
 \equiv & \hspace{10em} cx_1 = cx_1c, \text{ Lemma 4.2(ii) and } B = \bar{c}\bar{b}_1 \sqcap \bar{c}\bar{b}_2 \\
 & bcx_1(cb_1y_1 \sqcap \bar{c}b_2y_2)^\omega B \sqsubseteq bcx_1(cb_1y_1)^\omega c(\bar{c}\bar{b}_1 \sqcap \bar{c}\bar{b}_2) \\
 \equiv & \hspace{10em} c \text{ conjunctive, Lemma 4.2(ii)} \\
 & bcx_1(cb_1y_1 \sqcap \bar{c}b_2y_2)^\omega B \sqsubseteq bcx_1c(b_1y_1)^\omega \bar{b}_1
 \end{aligned}$$

$$\begin{aligned}
 & (bx)^\omega \bar{b}y \\
 & = \hspace{15em} y \text{ preserves } b \\
 & \bar{b}y \sqcap b(bx(\bar{b}y \sqcap b))^\omega \bar{b}.
 \end{aligned}$$

Set $A := b(bx(\bar{b}y \sqcap b))^\omega \bar{b}$. For the direct refinement,

$$\begin{aligned}
 & (bx)^\omega \bar{b}y \sqsubseteq \bar{b}y \sqcap A \\
 & \Leftarrow \hspace{15em} \text{Axiom (14), monotonicity of } \sqcap \\
 & bx(\bar{b}y \sqcap A) \sqsubseteq A \\
 & \equiv \hspace{15em} \text{using Axiom (13), } A = bx(\bar{b}y \sqcap b)(bx(\bar{b}y \sqcap b))^\omega \bar{b} \\
 & bx(\bar{b}y \sqcap A) \sqsubseteq bx(\bar{b}y \sqcap b)(bx(\bar{b}y \sqcap b))^\omega \bar{b} \\
 & \Leftarrow \hspace{15em} \text{monotonicity of } \wp, \text{ right distributivity (8)} \\
 & \bar{b}y \sqcap A \sqsubseteq \bar{b}y(bx(\bar{b}y \sqcap b))^\omega \bar{b} \sqcap A \\
 & \equiv \hspace{15em} b \text{ preserved by } y \\
 & \bar{b}y \sqcap A \sqsubseteq \bar{b}y\bar{b}(bx(\bar{b}y \sqcap b))^\omega \bar{b} \sqcap A \\
 & \equiv \hspace{15em} \bar{b}(bz)^\omega = \bar{b} \text{ (see below), } \bar{b}y\bar{b} = \bar{b}y, \text{ reflexivity of } \sqsubseteq \\
 & \text{true.}
 \end{aligned}$$

To show $\bar{b}(bz)^\omega = \bar{b}$, one simply unfolds $(bz)^\omega$ (Axiom (13)) and use the fact that guards are conjunctive.

For the reverse refinement, it has been shown in [11] page 115 that $bx(bx)^\omega \bar{b}y \sqsupseteq A$ and one deduces the result by monotonicity of \sqcap and unfolding Axiom (13). Solin's proof could be weakened requiring only Lemma 4.1(ii), conjunctivity of guards, left-subdistributivity, monotonicity of the operators and the preservation properties.

4.5 Transformation 4: composition

Consider the program

$$x_1 \wp \text{ do } b_1 \rightarrow y_1 \text{ od } \wp x_2 \wp \text{ do } b_2 \rightarrow y_2 \text{ od}.$$

Using the same reasoning as Kozen and Solin, the program x_1 can be ignored and considered as a part of the precomputation. Guard b_1 may be assumed to be conserved by x_2 and y_2 (using the technique of Transformation 3) so that x_2 can be absorbed into the body of the first loop. Therefore, it suffices to transform

Now, if x and y contain loops, it suffices to apply Transformation 1 and others as needed.

Observe that the expected number of steps to termination is preserved by reduction to single-loop normal form. Indeed the number of steps to termination of a loop is obtained by the value of a fresh variable that is initialized to 0 before the loop and incremented by 1 with each iteration (and can be arbitrarily bounded if the state space is to be finite). The expected number of iterations to termination is thus the expected value of that fresh variable, which is seen to be preserved, in the semantics, by the transformations to normal form. (Evidently the actual number of steps to termination need no longer -with presence of probabilism- be equal.)

5 Example: iterated random jumps

The purpose of this section is to study an example and in particular to demonstrate the invariance of the expected number of steps to termination. In general, interest lies in the extent to which the reasoning can be done, for the purposes of automation, within the algebra.

Consider a series of random jumps starting from some integer m and ending at the origin, with each jump made according to some state-dependent probability p_x (state x will be defined later). We are interested in the expected total number of jumps.

One such experiment is captured by nested loops in which the inner loop makes random jumps from m and the outer one decides whether or not to start new jumps:

```

c, x, y := 0, n, 0 ;
do x > 0 →
  (x := x-1) [p_x] (y := m ;
                    do y > 0 → RChoice(z, {1, ..., y}) ;
                    c, y := c+1, y-z
                  od)
od .

```

The procedure *RChoice* may be specified

```

proc RChoice(res z, val S)
  s :∈ S ;
  z := s [ $\frac{1}{\#S}$ ] RChoice(z, S \ {s})
corp .

```

The final value of c (the fresh variable mentioned at the end of the previous section) is the total number of jumps after an execution of *Prog*, so one associates to it a random variable T_n . One proves analytically that the normal form of this program has the same expected number of jumps $E(T_n)$.

But first, one calculates the expected value of the random variable J defined to be the number of jumps in one execution of the inner loop $Prog'$. Firstly, $Prog'$ performs random jumps from m to 0. It could be considered a Markov chain with associated matrix of transition probabilities M such that

$$M_{i,j} = \begin{cases} q_i & j < i \\ 0 & \text{otherwise} \end{cases}$$

where $q_i = 1/i$. Therefore

$$E(J) = \sum_{k \geq 0} k M_{m,0}^k.$$

This sum converges since M is nilpotent of order $m+1$. By induction on $k \geq 2$, one shows

$$M_{m,0}^k = q_m \sum_{1 \leq i_1 < \dots < i_{k-1} \leq m-1} q_{i_1} q_{i_2} \dots q_{i_{k-1}}.$$

By induction on m , $M_{m,0}^k$ is exactly the k^{th} coefficient of

$$F(x) = q_m x \prod_{i=1}^{m-1} (q_i x + 1)$$

i.e. $F(x)$ is the generating function of J . Now

$$\begin{aligned} & F'(1) \\ &= \\ & q_m [1 + \sum_{i=1}^{m-1} q_i / (q_i x + 1)] \prod_{j=1}^{m-1} (q_j x + 1) |_{x=1} \\ &= \\ & m^{-1} [1 + \sum_{i=1}^{m-1} 1/(1+i)] m! / (m-1)! \\ &= \\ & 1 + 1/2 + \dots + 1/m. \end{aligned} \qquad q_i = 1/i$$

Hence, $E(J) = H_m$, the m^{th} harmonic number.

Secondly, for $x = i$ ($1 \leq i \leq n$) denote by C_i the random variable defined to be the total number of executions of $Prog'$ followed by a decrease of x to the value $i-1$. C_i is geometrically distributed with probability p of success and therefore $E(C_i) = 1/p_i$. The random variable T_n becomes

$$T_n = \sum_{i \geq 1} (C_i - 1) J$$

Since the executions of $Prog'$ are independent, the expected total number of jumps is

$$E(T_n) = E(J) \sum_{i=1}^n (E(C_i) - 1) = H_m \sum_{i=1}^n \frac{\bar{p}_i}{p_i}.$$

When the p_i 's are equal and moreover constant with value p ,

$$E(T_n) = \frac{\bar{p} n H_m}{p}.$$

Now, let us focus on the normal form version of *Prog*. To simplify, One assumes $n > 0$ (the body of the outer loop is executed at least once). Applying transformations 5, 1, 2, using the hypothesis $n > 0$ and rearranging, the following normal-form program follows

```

c, x, y := 0, n, 0 ;
b, x := true, x-1 [p_x] b, y := false, m ;
do  $\bar{b}(y > 0) \sqcap (x > 0) \rightarrow$ 
  if  $\bar{b} \sqcap (y > 0) \rightarrow RChoice(z, \{1, \dots, y\}) ;$ 
    c, y := c+1, y-z
  []  $b \sqcap (y = 0) \rightarrow b, x := true, x-1 [p_x] b, y := false, m$ 
  fi
od

```

Noticing that $\bar{b} \sqsubseteq (y > 0)$, $y = 0 \sqsubseteq x = 0$ and that b is a fresh variable, one has

$$\bar{b}(y > 0) = (y > 0) \quad \text{and} \quad (y > 0) \sqcap (x > 0) = (x > 0)$$

and the normal form can be further simplified

```

c, x, y := 0, n, 0 ;
x := x-1 [p_x] y := m ;
do  $x > 0 \rightarrow$ 
  if  $y > 0 \rightarrow RChoice(z, \{1, \dots, y\}) ;$ 
    c, y := c+1, y-z
  []  $y = 0 \rightarrow x := x-1 [p_x] y := m$ 
  fi
od

```

Now, the structure of this loop is very similar to the original and so the random variables C_i 's, J and T_n may be used. The total number of jumps is given by the random variable

$$U = p_n T_{n-1} + \bar{p}_n (J + T_n)$$

and, using the linearity of expectation,

$$E(U) = H_m \sum_{i=1}^n \frac{\bar{p}_i}{p_i},$$

as required.

6 Conclusion

Kleene algebra provides a simple yet powerful formalism for reasoning about loops. With interest growing in probabilistic algorithms, it is natural to ask if Kleene algebra can be extended equally

successfully to probabilistic loops. However the interaction between probability and abstraction in the form of demonic nondeterminism is a subtle one. So the question becomes: how complex must be an extension of Kleene algebra in the presence of both probabilism and nondeterminism.

In this paper refinement algebra has been extended with two combinators for probabilistic choice: an unquantified choice and a partial quantified choice. Semantic models have been introduced to prove the axioms sound, and a folk theorem, extended to probabilistic programs, proved in order to demonstrate the strength of the axioms.

It is proposed, as further work, to study whether or not the equal asymptotic complexity of the two programs can be established by algebraic reasoning. It is also proposed to investigate what kind of algebraic reasoning requires further axioms involving the two forms of probabilistic choice.

Acknowledgment

The authors appreciate guidance and suggestions from Annabelle McIver.

References

- [1] E. Cohen. Separation and reduction. In R. Backhouse and J.N. Oliveira, editors, *Proceedings of the 5th International Conference on the Mathematics of Program Construction, MPC2000*, pages 45–59. Springer Verlag, 2000.
- [2] J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.
- [3] D. Harel. On folk theorems. *Communications of the ACM*, 23:379–389, 1980.
- [4] P. Höfner and G. Struth. Can refinement be automated? *Electronic Notes Theoretical Computer Science*, 201:197–222, 2008.
- [5] He Jifeng, A.K. McIver, and K. Seidel. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28:28–171, 1997.
- [6] D. Kozen. A completeness theorem for kleene algebras and the algebra of regular events. pages TR90–1123, 1990.
- [7] D. Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.
- [8] A.K. McIver, E. Cohen, and C.C. Morgan. Using probabilistic kleene algebra for protocol verification. 2006.

-
- [9] A.K. McIver and C.C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer Verlag, 2004.
 - [10] L.A. Meinicke. *Transformation Rules for Probabilistic Programs: An Algebraic Approach*. PhD thesis, The University of Queensland, Brisbane, Australia, 2008.
 - [11] K. Solin. *Abstract Algebra of Program Refinement*. PhD thesis, TUCS, Turku, Finland, 2007.
 - [12] D. Varacca and G. Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, 16(1):87–113, 2006.
 - [13] J. von Wright. From kleene algebra to refinement algebra. In *Proceedings of the 6th International Conference on the Mathematics of Program Construction, MPC'02*, pages 233–262, London, UK, 2002. Springer Verlag.