Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 201 (2008) 177–195
www.elsevier.com/locate/entcs

Refinement Algebra for Probabilistic Programs

Larissa Meinicke[1,3]

Åbo Akademi, Åbo, Finland

Kim Solin[2,4]

Åbo Akademi, Åbo, Finland

Abstract

We propose an abstract refinement algebra for reasoning about probabilistic programs in a total-correctness setting. The algebra is equipped with operators that determine whether a program is enabled, has certain failure or does not have certain failure, respectively. As an application, refinement rules for probabilistic action systems are derived in the algebra.

Keywords: refinement algebra, probability, Kleene algebra, action systems, data refinement

1 Introduction

Suppose one would like to replace the probabilistic program

$$\mathsf{while}\ g\ \mathsf{do}\ x\ {}_{p_1}\!\oplus\ y\ \mathsf{od};\mathsf{while}\ g\ \mathsf{do}\ u\ {}_{p_2}\!\oplus\ v\ \mathsf{od}$$

[1] Work partially done while at School of Information Technology and Electrical Engineering, University of Queensland. Partially supported by Australian Research Council (ARC) Discovery Grant DP0558408, *Analysing and generating fault-tolerant real-time systems.*
[2] Work partially done while visiting Institut für Informatik, Universität Augsburg.
[3] Email: larissa.meinicke@abo.fi
[4] Email: kim.solin@abo.fi

doi:10.1016/j.entcs.2008.02.020

by the probabilistic program

$$\mathsf{while}\ g\ \mathsf{do}\ x\ {}_{p_1}\!\oplus\ y\ \mathsf{od},$$

so as to get rid of the vacuous successor loop – the end goal being, for example, compiler optimisation. To do this, one would like a technique of transforming the initial program into the latter while preserving some notion of correctness. This technique should aspire to being both reliable and uncomplicated to use. Which methods are at hand?

Abstract algebra has a solid mathematical underpinning and simultaneously provides a perspicuous notation that allows for simple symbol pushing instead of tedious model-theoretic reasoning. There have been several examples of abstract algebra as an efficient reasoning tool. One of the earliest such abstract algebras is Kleene algebra with tests, which was employed by Kozen for proving transformation rules of loops [4]. Recently, McIver, Cohen and Morgan used probabilistic Kleene algebra for protocol verification [5] and Solin and von Wright used refinement algebras for program reasoning in a total-correctness environment [14,15,12,11].

In this paper, we present an abstract algebra for reasoning about probabilistic programs. It lifts the concrete-algebraic approach to probabilistic programs of Meinicke and Hayes [8] to a more abstract level, in the same way that Solin and von Wright [14,15,12] lift the concrete-algebraic approach to non-probabilistic programs of Back and von Wright [2]. The lifting not only provides a more perspicuous notation, but also allows for results proved in the algebra to be reused over different models for which the axiomatisation is sound. It also paves a more treadable way for automation.

We propose an abstraction which allows reasoning about total correctness, i.e. the program transformations that are done are valid also when the programs reasoned about are not necessarily terminating. The ability to assume that programs are possibly non-terminating is an important part of any form of reasoning about programs – a non-terminating loop is a classical programming error. Therefore a framework that *only* allows for reasoning in a partial correctness framework (such as the probabilistic Kleene algebra [5]) is not satisfactory.

The probabilistic refinement algebra has operators to represent sequential composition, choice and iteration. It is very similar to Kleene algebra, which constitutes a complete axiomatisation for the algebra of regular languages. The main differences to Kleene algebra is the addition of an iteration operator that is to model an iteration that either terminates or goes on forever, and the modification of three axioms. Although the elements in the carrier set may be interpreted to be probabilistic programs, we do not include a probabilistic

choice operator in our algebra. This decision reflects an important observation: many non-trivial transformation rules for probabilistic systems, such as the data refinement rules we derive in Sect. 6, may in fact be specified and verified without having to reason directly about probabilistic choices. We consider the elegance and simplicity of the framework to be a virtue. By not including a probabilistic choice operator we do not unnecessarily restrict its generality.

We do introduce guards and assertions into the refinement algebra. Guards form a Boolean subalgebra of the carrier set and can be used when modeling for example the predicates of conditionals and loops. Since Boolean algebra is again a very well-known structure, this does not endanger the simplicity of the abstraction. Assertions, which can also be used to model predicates but behave differently from guards when the predicate does not hold, are defined in terms of guards. Moreover, we define operators that determine whether a program is enabled, has certain failure (will abort with probability one) or does not have certain failure, respectively. The last-mentioned operators are similar to the domain operator of relational algebra and thus familiar to several with a background in computer science or discrete mathematics.

To show the elegance of the abstract-algebraic method, we apply the algebra to reasoning about action systems. Action systems can be used for modeling parallel or distributed systems in which concurrent behaviour is modeled by interleaving atomic actions [1]. Probabilistic action systems extend action systems to account also for probabilistic behaviour, in that the actions are allowed to be probabilistic programs [10]. Employing the enabledness and the certain-failure operator we show how to prove data refinement rules of probabilistic action systems in the refinement algebra.

Our contribution in this paper is two-fold: firstly, the construction of an abstract refinement algebra for probabilistic programs in the style of Solin and von Wright and the development of its basic theory, and secondly the algebra's application to the derivation of refinement rules for probabilistic action systems. To make the presentation clearer, we will first build up the theory and only when this is done we apply the algebra. In the two following sections we thus present a *probabilistic demonic refinement algebra*. We then introduce guards and assertions in Sect. 4, and enabledness and failure operators in Sect. 5. In Sect. 6 we use the refinement algebra to prove properties of probabilistic action systems: we derive three data refinement rules. We conclude in Sect. 7.

2 Probabilistic Demonic Refinement Algebra

In this section we introduce a probabilistic demonic refinement algebra. It abstracts the concrete expectation-transformer algebra of Meinicke and Hayes [8], and it is closely related to the abstract algebras presented in [15,5,13,9]. The algebra is axiomatised over the operators

$$;\,, \sqcap, ^*, \text{ and } ^\omega,$$

and the constants

$$\top \text{ and } 1.$$

The elements of the carrier set can be seen as probabilistic program statements. The operators should then be understood so that $\sqcap$ is *demonic choice* – a choice we cannot affect and which is not made with respect to any probability – and ; is *sequential composition*. The constant $\top$ is magic, a program statement that establishes any postcondition; and 1 is skip. *Weak iteration* * (the Kleene star) can be seen as an iteration of any finite length. *Strong iteration* $^\omega$ is an iteration that either terminates *or* goes on infinitely. The strong iteration operator may be used to model well-known programming statements such as while-loops, which are possibly non-terminating. While-loops that are certainly terminating may be more specifically modeled using the weak iteration operator. We also define a *refinement ordering* on the algebra by

$$x \sqsubseteq y \Leftrightarrow_{df} x \sqcap y = x$$

to be read "y establishes anything that x does and possibly more" (intuitively, if x is refined by y, then a demon would always choose x since y can do anything that x does and possibly more; by choosing x the demon has a better chance of winning).

Definition 2.1 *A* probabilistic demonic refinement algebra *(*pDRA*) is a structure over the signature*

$$(;\,, \sqcap, ^*, ^\omega, \top, 1)$$

satisfying the following axioms and rules ($\sqcap$ has least precedence, followed by ;, and then * and $^\omega$, which have equal precedence – we omit ; so that $x;y$ is written xy when no confusion can arise):

$$x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z, \tag{1}$$
$$x \sqcap y = y \sqcap x, \tag{2}$$
$$x \sqcap \top = x, \tag{3}$$
$$x \sqcap x = x, \tag{4}$$
$$x(yz) = (xy)z, \tag{5}$$
$$1x = x = x1, \tag{6}$$
$$\top x = \top, \tag{7}$$
$$x(y \sqcap z) \sqsubseteq xy \sqcap xz, \tag{8}$$
$$(x \sqcap y)z = xz \sqcap yz, \tag{9}$$
$$x^* = 1 \sqcap xx^*, \tag{10}$$
$$x^* = 1 \sqcap x^*(x \sqcap 1), \tag{11}$$
$$x \sqsubseteq yx \sqcap z \Rightarrow x \sqsubseteq y^* z, \tag{12}$$
$$x \sqsubseteq x(y \sqcap 1) \sqcap z \Rightarrow x \sqsubseteq zy^*, \tag{13}$$
$$x^\omega = 1 \sqcap xx^\omega \text{ and} \tag{14}$$
$$yx \sqcap z \sqsubseteq x \Rightarrow y^\omega z \sqsubseteq x, \tag{15}$$

where the order $\sqsubseteq$ *is defined by* $x \sqsubseteq y \Leftrightarrow_{df} x \sqcap y = x$. ◁

To model program abortion, we define a constant $\bot$ by

$$\bot =_{df} 1^\omega \tag{16}$$

and it is easily verified via axioms (15) and (6) that

$$\bot \sqsubseteq x \text{ and } \bot x = \bot \tag{17}$$

hold for any x [14], so $\bot$ is a least element and is right annihilating. In a program interpretation, $\bot$ may be seen as abort, a program establishing no postcondition. It is easy to prove that all the operators are isotone in all their arguments with respect to $\sqsubseteq$ and that $\sqsubseteq$ is a partial order.

The operators can be given an interpretation such that the set of expectation transformers over a fixed and finite state space that satisfy both sublinearity and infinite scaling forms a pDRA (cf. App. A). All the concepts introduced in the rest of this paper can also be given expectation-transformer interpretations as sketched in App. A (we will not explicitly point this out in the sequel). This interpretation is our motivating and working model, but it is not to be seen as exclusive.

Let us look a bit closer at some of the axioms. Most of those not pertaining to the strong iteration operator should be familiar from Kleene algebra: in fact one of the Kleene algebra axioms is absent and three have been modified. The annihilation axiom, $x\top = \top$, is absent so that we can model non-termination.

For the same reason it is also excluded from other algebras including the demonic refinement algebra [14] and Möller's lazy Kleene algebra [9]. The three axioms from Kleene algebra which have been modified are right distributivity,

$$x(y \sqcap z) = xy \sqcap xz, \tag{18}$$

and the unfolding and induction axioms

$$x^* = 1 \sqcap x^* x \text{ and} \tag{19}$$

$$x \sqsubseteq xy \sqcap z \Rightarrow x \sqsubseteq zy^*. \tag{20}$$

As noted by Meinicke and Hayes [8], these properties do not hold for programs which exhibit branching behaviour, and so right-distributivity is suitably weakened to right sub-distributivity (axiom (8)), and the unfolding and induction axioms are replaced by axioms (11) and (13). A similar generalisation is also made in probabilistic Kleene algebra [5] and the monodic tree Kleene algebra [13]. Axiom (11) may be derived from the others, but is included for clarity. [5]

As in the demonic refinement algebra (DRA) [14], we include unfolding (14) and induction (15) axioms for the strong iteration operator. Unlike the demonic refinement algebra and lazy omega algebra [9], our algebra does *not* include the isolation axiom, $x^\omega = x^* \sqcap x^\omega \top$, which states that a strong iteration may be decomposed into a simple choice between performing any finite or any infinite number of iterations: this property is not satisfied by probabilistic programs [8].

Although the primary focus of this paper is performing total correctness reasoning, we do not exclude discussions of the weak-iteration operator, since it can be applied when it is reasonable to assume that an iteration is terminating (see [5,8]). In Sect. 6 we show how weak iteration, via the new axiom (13), plays an important role in the derivation of a data refinement rule for probabilistic action systems.

3 Healthiness conditions and basic properties

In this section we introduce "healthiness conditions" and also derive some basic properties of the algebra. The healthiness conditions are stated in a fashion that would correspond to defining them in a *point-free way* in the expectation-transformer model, that is without going down to the level of expectations.

[5] In the monodic tree Kleene algebra the axiom $1 + x^*(x + 1) \leq x^*$ could thus also be elided.

We thus say that an element x is *conjunctive* if it satisfies

$$x(y \sqcap z) = xy \sqcap xz \tag{21}$$

for any y and z in the carrier set. As mentioned in the previous section, conjunctivity is not satisfied by all probabilistic programs, hence it is not an axiom of pDRA. It is, however, satisfied by a large subset of these: the programs which do not include probabilistic choices. When it is reasonable to take conjuctivity as an assumption, many useful transformation rules which would otherwise not hold, may be verified. Also, we say that an element x is *continuous* if the condition

$$(\forall n \bullet x(yn \sqcap u) \sqsubseteq zxn \sqcap vx) \Rightarrow xy^\omega u \sqsubseteq z^\omega vx \tag{22}$$

holds for any y, z, u and v in the carrier set (the bounded n is also in the carrier set). [6] Continuity is required in order to prove some useful commutativity rules for iterations.

Our axiomatisation extends von Wright's general refinement algebra [15] with axioms (11) and (13) concerning weak iteration, so every proposition proved in general refinement algebra also holds in pDRA. We do for example have the following decomposition and leapfrog properties of strong iteration.

Proposition 3.1 *For any x and y in the carrier set of a* pDRA

$$(x \sqcap y)^\omega = x^\omega (yx^\omega)^\omega, \tag{23}$$
$$x(yx)^\omega \sqsubseteq (xy)^\omega x \text{ and} \tag{24}$$
$$x(yx)^\omega = (xy)^\omega x, \text{ provided } x \text{ is conjunctive,} \tag{25}$$

hold.

In fact, the two first data refinement rules of Sect. 6 (in which we only consider strong iteration) are essentially applications of general refinement algebra – but taking into account the new definition of guards in the next section and the novel failure operator of Sect. 5. On the other hand, the new weak iteration axioms (11) and (13) are essential for the last data refinement rule of Sect. 6. The weak-iteration property

$$x^* = x^* x^*, \tag{26}$$

for any x in the carrier set of a pDRA, is easy to prove and will be used in our application.

[6] The condition is closely related to the fusion lemma of fixpoint theory. So, continuity is, in a certain sense, here defined *via* the fusion lemma. A point-free abstraction of the common definition of continuity would require an angelic-choice operator.

4 Guards, Assertions and the Initial Question

We now introduce guards and assertions into the algebra. Guards are to be seen as statements that check if a predicate holds and skip if the predicate holds, otherwise behave like magic. Guards must be introduced slightly differently than in [14,15], since not every element is conjunctive but we still want guards to satisfy conjunctivity. Hence, an element g of the carrier set that

- is conjunctive and
- has a conjunctive complement $\bar{g}$ satisfying $\bar{g}g = g\bar{g} = \top$ and $g \sqcap \bar{g} = 1$

is called a *guard*. The first guard equation in the second condition says that either a predicate or its negation holds, and therefore a sequential composition of a guard and its complement will always result in a miracle. The second guard equation says that a demon will always be able to make the program skip when choosing between a guard and the guard's complement. It can be established that the guards form a Boolean algebra (BA) over $(\sqcap, ;, \bar{}, 1, \top)$, where $\sqcap$ is meet, ; is join, $\bar{}$ is complement, 1 is the least element, and $\top$ is the greatest element [14].

Every guard is defined to have a corresponding *assertion*

$$g^\circ = \bar{g}\bot \sqcap 1 \tag{27}$$

and so $^\circ$ is a mapping from guards to a subset of the carrier set: the set of assertions. Assertions work in the same way as guards, except that they abort if the predicate does not hold. If the predicate does not hold, then a demon would choose the left hand side of the demonic choice: the negated guard would skip and the whole program abort (which is what a demon wants). If, on the other hand, the predicate holds, then a demon would choose the right hand side, since otherwise the negated guard would do magic and the demon would lose (the demon could then no longer establish abortion). Note that

$$g^\circ = \bar{g}\bot \sqcap g \tag{28}$$

can be shown equivalent to (27). It is easy to show that

$$g_1^\circ \sqsubseteq 1 \sqsubseteq g_2 \tag{29}$$

holds for any assertion g_1° and any guard g_2 [14]. The properties

$$gg^\circ = g \text{ and } g^\circ g = g^\circ \tag{30}$$

are also easy to prove and will be used later on.

As already mentioned, pDRA specifically requires that guards are conjunctive, which differs from DRA in which *all* elements by axiomatisation are conjunctive. On the other hand, the conjunctivity of assertions can be shown (by the definition of assertions and applying some basic axioms) to follow from the conjunctivity of guards.

Consider now our initial question in the introduction. With guards defined, we can express, for example, a probabilistic while-loop

$$\mathsf{while}\ g\ \mathsf{do}\ x\ {}_p\!\oplus\ y\ \mathsf{od}$$

as $(gu)^\omega \bar{g}$ in the algebra, where u is the abstraction of the binary probabilistic choice between probabilistic programs x and y, and the guard g expresses the guard of the loop. Our initial question can then be rephrased to asking if

$$(gu)^\omega \bar{g}; (gv)^\omega; \bar{g} = (gu)^\omega; \bar{g}$$

is true for any guard g and any u and v in the carrier set of a pDRA. That this is indeed the case follows from that fact that for any guard g and any carrier-set element x, it can be proved that

$$\bar{g}(gx)^\omega = \bar{g}$$

holds (by unfolding of strong iteration (14), the definition of guards and the fact that $\top$ is the top element (3)). This and the fact that the guards form a Boolean algebra shows that the successor loop can be elided.

5 Enabledness and Failure

In this section we introduce the enabledness and the failure operators. The enabledness operator behaves just like in refinement algebra for non-probabilistic programs and the not-certain-failure operator is a probabilistic counterpart to the termination operator of non-probabilistic refinement algebra [12].

5.1 Enabledness

The *enabledness operator* ϵ is a unary operator that maps a carrier-set element of a pDRA to a guard and satisfies the axioms

$$\epsilon x x = x, \tag{31}$$
$$g \sqsubseteq \epsilon(gx), \tag{32}$$
$$\epsilon(xy) = \epsilon(x\epsilon y) \text{ and} \tag{33}$$
$$\epsilon x \bot = x \bot. \tag{34}$$

We will call a pDRA with an enabledness operator a pDRAe.

The intuition of ϵx is that it returns a guard that skips in those states from which x is not miraculous. That is to say, ϵx checks whether the program is enabled or not. The first axiom thus says that a program is equal to the same program preceded by a guard that checks if it is enabled: if the program is enabled, the guard skips and then executes the program, if the program is not enabled (that is, it will perform a miracle), the guard will not hold and thus the whole program will do magic. The other axioms can be interpreted similarly. Our axioms are suitable for probabilistic models in which programs are either miraculous from a given initial state, or they are not.

The properties

$$\epsilon(x \sqcap y) = \epsilon x \sqcap \epsilon y \text{ and} \tag{35}$$

$$x \sqsubseteq y \Rightarrow \epsilon x \sqsubseteq \epsilon y \tag{36}$$

can be shown to hold by a similar proof as for the domain operator in [3], taking into consideration that guards are conjunctive.

5.2 Termination and Failure

For non-probabilistic demonic refinement algebra, a termination operator was defined by Solin and von Wright [12,11]. When applied to a non-probabilistic program, this termination operator returns an assertion that skips in those states from which the program will terminate, otherwise it aborts.

Since probabilistic programs terminate *with some probability*, a probabilistic termination operator might be expected to return a *probabilistic assertion* that skips with a probability equal to the termination probability of the program and aborts with a probability equal to the probability that the program will not terminate. The usefulness of such a termination operator is debatable, and, to express it we would also have to introduce probabilistic assertions into our algebra. Instead, we define a *not-certain-failure operator* which acts as a counterpart to the non-probabilistic termination operator. We define the not-certain-failure operator via an abstraction of the *fail operator* defined on expectation transformers by Meinicke and Hayes [8]. The intuition behind the fail operator is a guard that checks whether a program certainly aborts, and the intuition behind the not-certain-failure operator an assertion that skips in those states from which the program does not certainly fail – that is, the program has got a chance of terminating.

We denote the fail operator by f and define it as a mapping from the carrier set to the set of guards satisfying

$$\mathsf{f}x\bot = x\top. \tag{37}$$

A pDRA with a failure operator we will denote pDRAf and a pDRA with an

enabledness operator and a failure operator we denote pDRAef. [7]

When we define the *not-certain-failure operator* τ as

$$\tau x = \overline{\mathsf{f}x}^{\circ} \tag{38}$$

this operator satisfies the axioms of the termination operator of non-probabilistic demonic refinement algebra [12,11]. That is to say, it can be verified (directly from (38) and axiom (37)) that τ satisfies

$$x = \tau x x, \tag{39}$$
$$\tau(g^{\circ}x) \sqsubseteq g^{\circ}, \tag{40}$$
$$\tau(x\tau y) = \tau(xy), \tag{41}$$
$$\tau x \top = x\top \text{ and} \tag{42}$$
$$\tau(x \sqcap y) = \tau x \sqcap \tau y. \tag{43}$$

Let us again look at the first axiom. It says that any program x is equal to a program consisting of an assertion that first checks that x will not certainly fail (will not abort) and then executes x: if x certainly fails, then the assertion fails so the composite program aborts, whereas if x does not certainly fail, then the assertion skips and x gets executed. Note that the fail operator cannot be defined in terms of the not-certain-failure operator, since this would demand the possibility to define guards in terms of assertions.

The failure operator satisfies the following properties that we will use later on. The first property says that a program can be refined by statement that checks that the program fails and then aborts (remember that the failure is checked by a guard, and thus a miracle happens if the program fails). The second property says that any program composed of any number of statements is refined by a program that checks if the first statement fails.

Proposition 5.1 *For any x and y in the carrier set of a* pDRAef

$$x \sqsubseteq \mathsf{f}x\bot \text{ and} \tag{44}$$
$$xy \sqsubseteq \mathsf{f}x \tag{45}$$

hold.

Proof. The first part is proved by

$$x = x(\top \sqcap 1) \sqsubseteq x\top \sqcap x = \mathsf{f}x\bot \sqcap x,$$

which holds by the definition of $\sqsubseteq$, the basic axioms (8) and (6) of the algebra and axiom (37). Then, by the first part of this proposition (44), property (17)

[7] It can be shown that not all pDRAs are pDRAefs.

twice and axiom (6) we have

$$xy \sqsubseteq \mathsf{f}x\bot y = \mathsf{f}x\bot \sqsubseteq \mathsf{f}x1 = \mathsf{f}x$$

which proves the second property. □

6 Data Refinement of Probabilistic Action Systems

This section comprises an application of our algebra to data refinement of action systems. Action systems can be used for reasoning about parallel or distributed systems in which concurrent behaviour is modeled by interleaving atomic actions [1]. Probabilistic action systems extend action systems to account also for probabilistic behaviour, in that the actions are allowed to be probabilistic programs [10]. An action system

$$\mathsf{do}\, x_1 [\!] \ldots [\!] x_n \,\mathsf{od}$$

is an iteration of a set of *actions* $x_1, ..., x_n$ that terminates when none of the actions are enabled, that is to say, the iteration continues as long as any action is enabled. In the abstract algebra, we encode an action system as a strong iteration of a demonic choice between n actions and we express the termination condition with the aid of the enabledness operator [12,2,8]:

$$\mathsf{do}\, x_1 [\!] \ldots [\!] x_n \,\mathsf{od} =_{df} (x_1 \sqcap \cdots \sqcap x_n)^{\omega} \overline{\epsilon x_1} \ldots \overline{\epsilon x_n}.$$

The strong-iteration operator allows us to model action systems in tune with our urge for total-correctness: we allow infinite iterations to be expressed.

During the derivation of a program, the process of refining a so-called abstract program by a so-called concrete program that uses a different data representation is called data refinement. For probabilistic programs y, z and x, the program y is said to be *data refined* by z through x if either

$$xy \sqsubseteq zx \text{ or } yx \sqsubseteq xz.$$

In the first instance, the probabilistic program x can be seen to represent a (probabilistic) mapping from the concrete state of z to the abstract state of y, and in the second x can be seen to represent a (probabilistic) mapping from the abstract state of y to the concrete state of z. We refer to data refinement in the first instance as *upward simulation*, and *downward simulation* in the second instance.

Upward simulation. An upward simulation data-refinement rule for probabilistic action systems

$$x; \mathsf{do}\ y\ \mathsf{od} \sqsubseteq \mathsf{do}\ z\ \mathsf{od}; x$$

can be shown to hold assuming that

$$\begin{aligned} xy &\sqsubseteq zx \text{ and} \\ x(\mathsf{f}y \sqcap \overline{\epsilon y}) &\sqsubseteq \overline{\epsilon z} x \end{aligned}$$

hold, that is, assuming that y is upwards data refined by z through x, and assuming that $\mathsf{f}y \sqcap \overline{\epsilon y}$ is upwards data refined by $\overline{\epsilon z}$ through x. The first condition constrains the loop body y to be data refined by z, the second constrains the termination of the loops: it states that z may only be disabled when y either aborts or is disabled. To prove this we first derive an important general commutativity property: for any x, y, z, u and v in the carrier set of a pDRAef such that x is continuous we have that

$$xy^\omega u \sqsubseteq z^\omega v x \tag{46}$$

provided

$$xy \sqsubseteq zx \text{ and} \tag{47}$$

$$x(\mathsf{f}y \sqcap u) \sqsubseteq vx. \tag{48}$$

It can be proved as follows. For any n,

$$\begin{aligned} & x(yn \sqcap u) \\ = \quad & \{\![\text{ idempotence (4) }]\!\} \\ & x(yn \sqcap u) \sqcap x(yn \sqcap u) \\ \sqsubseteq \quad & \{\![\text{ definition of } \sqsubseteq, \text{ (45), isotony }]\!\} \\ & xyn \sqcap x(\mathsf{f}y \sqcap u) \\ \sqsubseteq \quad & \{\![\text{ assumptions (47) and (48), isotony }]\!\} \\ & zxn \sqcap vx, \end{aligned}$$

and since x is continuous, the desired property follows from the continuity condition (22). The data refinement rule then follows by setting u to be $\overline{\epsilon y}$ and v to be $\overline{\epsilon z}$.

The above commutativity property is more general than the theorem proved by Meinicke and Hayes in the concrete expectation transformer algebra [8], since there u and v are constrained to be guards.

Downward simlulation. Probabilistic action systems also have a downward simulation data-refinement rule, given by

$$\mathsf{do}\ y\ \mathsf{od}; x \sqsubseteq x; \mathsf{do}\ z\ \mathsf{od}$$

provided

$$(\epsilon y)^\circ yx \sqsubseteq x(\epsilon z)^\circ z \text{ and}$$
$$x\overline{\epsilon z} \sqsubseteq \overline{\epsilon y}x$$

hold, that is, provided $(\epsilon y)^\circ y$ is downwards data refined by $(\epsilon z)^\circ z$ through x, and provided $\overline{\epsilon z}$ is upwards data refined by $\overline{\epsilon y}$ through x. To prove this, we again first establish a general commutativity property: for any x, y and z in the carrier set and g_1 and g_2 in the guard set of a pDRA we have that

$$(g_1 y)^\omega \bar{g}_1 x \sqsubseteq x(g_2 z)^\omega \bar{g}_2 \tag{49}$$

provided

$$g_1{}^\circ yx \sqsubseteq x g_2{}^\circ z \text{ and} \tag{50}$$
$$x\bar{g}_2 \sqsubseteq \bar{g}_1 x. \tag{51}$$

This property was first proved in the concrete expectation-transformer algebra by Meinicke and Hayes [8]. We here present their proof as a smooth and transparent derivation in the abstract algebra: First, we have that

$$\begin{array}{ll} & (g_1 y)^\omega \bar{g}_1 x \sqsubseteq x(g_2 z)^\omega \bar{g}_2 \\ \Leftarrow & \quad \{\!\![\text{ (15) }]\!\!\} \\ & g_1 yx(g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_1 x \sqsubseteq x(g_2 z)^\omega \bar{g}_2 \end{array}$$

holds. That the antecedent follows from the assumptions is settled by

$$\begin{array}{ll} & g_1 yx(g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_1 x \\ = & \quad \{\!\![\text{ (30) }]\!\!\} \\ & g_1 g_1{}^\circ yx(g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_1 x \\ \sqsubseteq & \quad \{\!\![\text{ assumption (50), isotony }]\!\!\} \\ & g_1 x g_2{}^\circ z(g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_1 x \\ = & \quad \{\!\![\text{ assertion definition (28) }]\!\!\} \\ & g_1 x(\bar{g}_2 \bot \sqcap g_2) z(g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_1 x \\ = & \quad \{\!\![\text{ left distributivity (9) and preemption (17) }]\!\!\} \\ & g_1 x(\bar{g}_2 \bot \sqcap g_2 z(g_2 z)^\omega \bar{g}_2) \sqcap \bar{g}_1 x \\ \sqsubseteq & \quad \{\!\![\bot \sqsubseteq 1, \text{ commutativity (2) }]\!\!\} \\ & g_1 x(g_2 z(g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_2) \sqcap \bar{g}_1 x \\ = & \quad \{\!\![\text{ left distributivity (9) }]\!\!\} \\ & g_1 x(g_2 z(g_2 z)^\omega \sqcap 1)\bar{g}_2 \sqcap \bar{g}_1 x \\ = & \quad \{\!\![\text{ folding (14) }]\!\!\} \\ & g_1 x(g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_1 x \\ \sqsubseteq & \quad \{\!\![1 \text{ is unit (6), (29) }]\!\!\} \\ & g_1 x(g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_1 x \bar{g}_2 \\ = & \quad \{\!\![\text{ guards BA, preemption (7),} \top \text{ is unit (3) }]\!\!\} \\ & g_1 x(g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_1 x(\bar{g}_2 g_2 z(g_2 z)^\omega \sqcap \bar{g}_2) \end{array}$$

$$
\begin{array}{ll}
= & \quad \{\![\text{ guards are conjunctive }]\!\} \\
& g_1 x (g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_1 x \bar{g}_2 (g_2 z (g_2 z)^\omega \sqcap 1) \\
\sqsubseteq & \quad \{\![\text{ folding (14), 1 is unit (6), (29) }]\!\} \\
& g_1 x (g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_1 x \bar{g}_2 (g_2 z)^\omega \bar{g}_2 \\
\sqsubseteq & \quad \{\![\text{ assumption (51), guards BA }]\!\} \\
& g_1 x (g_2 z)^\omega \bar{g}_2 \sqcap \bar{g}_1 x (g_2 z)^\omega \bar{g}_2 \\
= & \quad \{\![\text{ left distributivity (9) }]\!\} \\
& (g_1 \sqcap \bar{g}_1) x (g_2 z)^\omega \bar{g}_2 \\
= & \quad \{\![\text{ guard properties }]\!\} \\
& x (g_2 z)^\omega \bar{g}_2 .
\end{array}
$$

The data refinement rule then follows by setting g_1 to be ϵy, setting g_2 to be ϵz, and taking the first enabledness axiom (31) into account.

General downward simulation. Unlike both the upward and downward simulation rules that have been presented, the following simulation rule allows data refinements to be verified between action systems in which finite sequences of *stuttering steps* have been added or removed – so a direct correspondence between the actions is not required. Since the sequences of stuttering steps are assumed to be finite, the verification of this rule requires reasoning about both weak and strong iterations. Importantly, the finite-iteration assumption allows us to use the new weak iteration axiom (13), for which there is no counterpart for strong iteration. The rule [7] states that

$$\mathsf{do}\ y\ \mathsf{od}; x \sqsubseteq x; \mathsf{do}\ z\ \mathsf{od}$$

provided $y = y_\natural \sqcap y_\sharp$, $z = z_\natural \sqcap z_\sharp$, ${y_\natural}^\omega = {y_\natural}^*$, ${z_\natural}^\omega = {z_\natural}^*$, and

$$
\begin{array}{rl}
{y_\natural}^* x \sqsubseteq x(z_\natural \sqcap 1), & (52) \\
(\epsilon y)^\circ y_\sharp {y_\natural}^* x \sqsubseteq x (\epsilon z)^\circ z_\sharp \text{ and} & (53) \\
x \overline{\epsilon z} \sqsubseteq \overline{\epsilon y} x. & (54)
\end{array}
$$

We refer to $y_\natural$ and $z_\natural$ as the stuttering actions, and $y_\sharp$ and $z_\sharp$ as the non-stuttering actions. The proof of Meinicke and Hayes [7] may then be expressed as a derivation in our abstract algebra.

From decomposition (Prop. 3.1 (23)), the general commutativity property presented for downward simulation, and the assumption that the iterations of $y_\natural$ and $z_\natural$ are terminating, it is sufficient to show that the following conditions hold

$$
\begin{array}{rl}
{y_\natural}^* x \sqsubseteq x {z_\natural}^*, & (55) \\
(\epsilon y)^\circ y_\sharp {y_\natural}^* x \sqsubseteq x (\epsilon z)^\circ z_\sharp {z_\natural}^* \text{ and} & (56) \\
x \overline{\epsilon z} \sqsubseteq \overline{\epsilon y} x. & (57)
\end{array}
$$

Condition (55) may be shown to hold given (52) as follows:

$$
\begin{array}{ll}
 & y_\natural{}^* x \sqsubseteq x(z_\natural \sqcap 1) \\
\Rightarrow & \quad \{\![\text{ isotony }]\!\} \\
 & y_\natural{}^* y_\natural{}^* x \sqsubseteq y_\natural{}^* x(z_\natural \sqcap 1) \\
\Leftrightarrow & \quad \{\![(26)]\!\} \\
 & y_\natural{}^* x \sqsubseteq y_\natural{}^* x(z_\natural \sqcap 1) \\
\Rightarrow & \quad \{\![\text{ from unfolding (10) } y_\natural{}^* x \sqsubseteq x, \text{ and refinement definition }]\!\} \\
 & y_\natural{}^* x \sqsubseteq y_\natural{}^* x(z_\natural \sqcap 1) \sqcap x \\
\Rightarrow & \quad \{\![\text{ induction (11) }]\!\} \\
 & y_\natural{}^* x \sqsubseteq x z_\natural{}^*
\end{array}
$$

Assuming (55) and (53), (56) can be shown to hold by

$$
\begin{array}{ll}
 & (\epsilon y)^\circ y_\sharp y_\natural{}^* x \\
= & \quad \{\![(26)]\!\} \\
 & (\epsilon y)^\circ y_\sharp y_\natural{}^* y_\natural{}^* x \\
\sqsubseteq & \quad \{\![\text{ assumption (55) }]\!\} \\
 & (\epsilon y)^\circ y_\sharp y_\natural{}^* x z_\natural{}^* \\
\sqsubseteq & \quad \{\![\text{ assumption (53) }]\!\} \\
 & x(\epsilon z)^\circ z_\sharp z_\natural{}^*.
\end{array}
$$

Conditions (57) and (54) are equal.

7 Concluding Remarks and Outlook

In this paper we showed that abstract-algebraic reasoning also works well when probabilistic programs are concerned. We proposed a probabilistic refinement algebra facilitating total correctness and including a strong iteration operator, as well as operators for enabledness and certain failure. As an application, probabilistic action systems were cast in the abstract algebra and three data refinement rules were established.

This research could be taken further in several directions. One direction would be to introduce angelic choice into the algebra and another direction would be to consider probabilistic choice abstractly. It should also be interesting to apply the algebra to more elaborate case studies.

Acknowledgement

The authors are grateful to R.J.R. Back, Ian J. Hayes, Bernhard Möller, Graeme Smith and an anonymous referee for helpful suggestions.

References

[1] R.J.R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. In *Proc. of the 2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, pages 131–142. ACM Press, 1983.

[2] R.J.R. Back and J. von Wright. Reasoning algebraically about loops. *Acta Informatica*, 36(4):295–334, 1999.

[3] J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM Transactions on Computational Logic*, 7(4):798–833, 2006.

[4] Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, 1997.

[5] A. K. McIver, E. Cohen, and C. C. Morgan. Using probabilistic Kleene algebra for protocol verification. In *Relations and Kleene Algebra in Computer Science*, volume 4136 of *LNCS*, pages 296–310, 2006.

[6] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005.

[7] Larissa Meinicke and Ian J. Hayes. Algebraic reasoning for probabilistic action systems and while-loops. Technical Report SSE-2006-05, School of Information Technology and Electrical Engineering, The University of Queensland, September 2006.

[8] Larissa Meinicke and Ian J. Hayes. Reasoning algebraically about probabilistic loops. In *Eighth Internatinal Conference on Formal Engineering Methods*, volume 4260 of *LNCS*, 2006.

[9] Bernhard Möller. Lazy Kleene algebra. In *Mathematics of Program Construction*, volume 3125 of *LNCS*, pages 252–273. Springer-Verlag, 2004.

[10] Kaisa Sere and Elena Troubitsyna. Probabilities in action systems. In *Proc. of the 8th Nordic Workshop on Programming Theory*. Helsinki: Publishing Association, 1996.

[11] Kim Solin. On two dually nondeterministic refinement algebras. In *Relations and Kleene Algebra in Computer Science*, volume 4136 of *LNCS*, pages 373–387, 2006.

[12] Kim Solin and Joakim von Wright. Refinement algebra with operators for enabledness and termination. In *Mathematics of Program Construction*, volume 4014 of *LNCS*, pages 397–415. Springer, 2006.

[13] Toshinori Takai and Hitoshi Furusawa. Monodic tree Kleene algebra. In *Relations and Kleene Algebra in Computer Science*, volume 4136 of *LNCS*, pages 402–416, 2006.

[14] Joakim von Wright. From Kleene algebra to refinement algebra. In *Mathematics of Program Construction*, volume 2386 of *LNCS*, pages 233–262. Springer, 2002.

[15] Joakim von Wright. Towards a refinement algebra. *Science of Computer Programming*, 51:23–45, 2004.

A Expectation transformers

This short appendix is written for those who are already somewhat familiar with the theory of expectation transformers. It provides an expectation-transformer interpretation of the concepts involved in the probabilistic refinement algebra. This appendix can unfortunately not be completely self-contained and self-explanatory, but we refer the interested reader to the comprehensive book by McIver and Morgan [6]. We use a slight variation

of the original theory of expectation transformers [6] so that miraculous program behaviour can be expressed. In this variation, which is documented in detail in [8], probabilistic programs may be miraculous from a given initial state with probability either 0 or 1.

Let p be a predicate on state space Σ; ϕ be an expectation of type $\mathcal{E}\Sigma$ $(= \Sigma \to \mathbb{R}_{\geq 0} \cup \{\infty\})$; and S and T be expectation transformers of type $\mathcal{E}\Sigma \to \mathcal{E}\Sigma$. Let c_1 and c_2 be constants of type $\mathbb{R}_{\geq 0} \cup \{\infty\}$, c be a constant of type $\mathbb{R}_{\geq 0}$, β_1, β_2 and β be expectations, and $\mathcal{B}$ be a directed set of expectations. For any subset $\mathcal{B}$ of a partially ordered set $\mathcal{A}$, $\mathcal{B}$ is *directed* if $(\forall \alpha, \beta \in \mathcal{B} \bullet (\exists \gamma \in \mathcal{B} \bullet \alpha \sqsubseteq \gamma \wedge \beta \sqsubseteq \gamma))$ [2].

magic	$\top.\phi$	$\mathsf{Infinity} \quad (= (\lambda\sigma \in \Sigma \bullet \infty))$
skip	$1.\phi$	ϕ
sequential composition	$(S;T).\psi$	$S.(T.\psi)$
demonic choice	$(S \sqcap T).\psi$	$S.\psi \sqcap T.\psi$
weak iteration	$(S^*).\phi$	$(\nu X \bullet S; X \sqcap 1).\phi$
strong iteration	$(S^\omega).\phi$	$(\mu X \bullet S; X \sqcap 1).\phi$
abort	$\bot.\phi$	$\mathsf{False} \quad (= (\lambda\sigma \in \Sigma \bullet 0))$
guard	$[p].\phi$	$(\lambda\sigma \in \Sigma \bullet \mathsf{if}\ p.\sigma\ \mathsf{then}\ \phi.\sigma\ \mathsf{else}\ \infty)$
assertion	$\{p\}.\phi$	$(\lambda\sigma \in \Sigma \bullet \mathsf{if}\ p.\sigma\ \mathsf{then}\ \phi.\sigma\ \mathsf{else}\ 0)$
enabledness	$[\epsilon.S]$	$[(\lambda\sigma \in \Sigma \bullet S.\mathsf{False}.\sigma = 0)]$
certain failure	$[\mathsf{f}.S]$	$[(\lambda\sigma \in \Sigma \bullet S.\mathsf{Infinity}.\sigma = 0)]$

$$(c_1 * S.(\beta_1) + c_2 * S.(\beta_2)) \ominus c \leq S.((c_1 * \beta_1 + c_2 * \beta_2) \ominus c) \quad \text{(sublinearity)}$$

$$S.(\infty * \beta) = \infty * S.\beta \quad \text{(infinite scaling)}$$

$$S.\beta_1 \sqcap S.\beta_2 = S.(\beta_1 \sqcap \beta_2) \quad \text{(conjunctivity)}$$

$$S.(\sqcup\beta \in \mathcal{B} \bullet \beta) = (\sqcup\beta \in \mathcal{B} \bullet S.\beta) \quad \text{(continuity)}$$

Fig. A.1. Expectation transformer interpretation and healthiness conditions for expectation transformers.

In Fig. A.1 the operators, constants and special elements that were treated in the paper are listed in the order of their introduction, and given an expectation transformer interpretation. Fig. A.1 also lists the expectation-transformer concepts that correspond to the healthiness conditions that were mentioned in the paper, and it also contains the definitions of sublinearity and infinite scaling. Informally, infinite scaling and sublinearity characterise the set of probabilistic programs that may include discrete probabilistic and demonic choice. As mentioned, the set of sublinear and infinitely scaling expectation transformers over a fixed and finite state space form a pDRAef.