DOI 10.1007/s00165-009-0111-1
BCS © 2009
Formal Aspects of Computing (2010) 22: 3–31

Formal Aspects of Computing

Refinement algebra for probabilistic programs

Larissa Meinicke[1] and Kim Solin[2]

[1] Åbo Akademi, Åbo, Finland. E-mail: larissa.meinicke@abo.fi
[2] Uppsala University, Uppsala, Sweden. E-mail: kim.solin@filosofi.uu.se

Abstract. We identify a refinement algebra for reasoning about probabilistic program transformations in a total-correctness setting. The algebra is equipped with operators that determine whether a program is enabled or terminates respectively. As well as developing the basic theory of the algebra we demonstrate how it may be used to explain key differences and similarities between standard (i.e. non-probabilistic) and probabilistic programs and verify important transformation theorems for probabilistic action systems.

Keywords: Refinement algebra, Probability, Kleene algebra, Action systems, Data refinement, Atomicity refinement

1. Introduction

Probabilistic programs, programs in which probabilistic choices may be made, have applications in areas such as distributed computing and reliable systems modelling. In order to be able to reason about such programs one would like a technique for understanding how and when it is possible to transform one probabilistic program into another while preserving some notion of correctness. For example, suppose we have a program

$$\mathsf{do}\ e \sqcap (a\ \mathsf{do}\ b\ \mathsf{od}) \sqcap l \sqcap r\ \mathsf{od} \tag{1}$$

in which e, a, b, l and r represent programs which may include discrete probabilistic choices (we use $x\ {}_p\!\oplus\ y$ to denote the discrete probabilistic choice in which x is executed with probability p and y is taken with probability $1 - p$), in addition to the more commonplace operators sequential composition, ; , and nondeterministic choice, $\sqcap$, which may be used to represent design freedom in specifications, or uncertainty. These programs may implicitly contain a guard, which denotes when they are enabled to be executed. Program (1), which we shall refer to as a *probabilistic action system* [ST96], may be used to represent the concurrent execution of atomic actions, e, (a do b od), l and r by an unfair scheduler: on each iteration the scheduler nondeterministically selects an enabled action for execution; it continues execution indefinitely, or until all of the actions become disabled. We may like to know under what circumstances is it possible to replace Program (1) by another, say

$$\mathsf{do}\ e \sqcap a \sqcap b \sqcap l \sqcap r\ \mathsf{od} \tag{2}$$

in which atomic action (a do b od) has been decomposed, thereby increasing the amount of parallelism in the system. Theorems for reasoning about transformations of this kind are often referred to as *separation and reduction* or *atomicity refinement* theorems, and have been shown to play a useful role in the development and

Correspondence and offprint requests to: L. Meinicke and K. Solin, E-mail: larissa.meinicke@abo.fi; kim.solin@filosofi.uu.se

verification of non-probabilistic distributed systems [Doe77, Lip75, LS89, Bac89, BvW99, Coh00]. We may also be interested in knowing when, and in what context, Program (2) may be replaced by replaced by another

$$\mathsf{do}\ e' \sqcap a' \sqcap b' \sqcap l' \sqcap r'\ \mathsf{od}$$

which uses a different data representation. Theorems of this variety, *data refinement* theorems, may be used to develop abstract program specifications into concrete program implementations.

The answers to these questions are not as straightforward and simple as they may seem. Although transformation rules for non-probabilistic programs are well understood, transformation rules for probabilistic programs differ in subtle, but important ways from their better-known non-probabilistic counterparts [MW05, MCM06, MH06, MH08b, Mei08]. A technique for justifying and explaining such transformations 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. Several examples have demonstrated how abstract algebra may be employed as an efficient tool for reasoning about programs. For example, in early work, Kozen used *Kleene algebra with tests* for proving transformation rules of loops in a partial-correctness framework [Koz97]. Solin and von Wright have then used *refinement algebras* for program reasoning in a total-correctness environment [vW02, vW04, SvW06, Sol06]. In addition to providing a suitable level of abstraction at which to perform and explain proofs, such algebras are a convenient way to describe similarities and dissimilarities between different program models and to reuse results proved in an algebra over different models for which the axiomatisation is sound. Recent results have also shown that abstract-algebraic proofs may be simple to automate [HS08].

McIver et al. [MW05, MCM06] and Meinicke and Hayes [MH06, MH08b, Mei08] have identified that algebraic reasoning also works well where probabilistic programs are concerned. In their work, McIver et al. [MW05, MCM06] specified a relaxation of Kleene algebra suitable for probabilistic programs, known as *probabilistic Kleene algebra*, and used it to derive a probabilistic separation and reduction theorem which has been applied in the verification of a protocol in [MCM06]. This algebra, like Kleene algebra, contains a weak iteration (or *Kleene star*) operator, *, which may be used to represent finite (or terminating) iterations, but it does not contain operators for representing possibly infinite iterations. For this reason, we say that it is suitable for reasoning about partial—but not total—program correctness. In related work, Meinicke and Hayes [MH06, MH08b, Mei08] explored algebraic properties of probabilistic programs within a total-correctness framework, in which possibly infinite iterations are expressible, and used them to derive transformation theorems for probabilistic loops and action systems within a particular program model.

In this paper, we identify and explore a very general abstract algebra for reasoning about probabilistic programs in a total-correctness framework. That is, we lift the concrete-algebraic approach to probabilistic programs of Meinicke and Hayes [MH06, MH08b] to a more abstract level, in the same way that Solin and von Wright [vW02, vW04, SvW06] lifted the concrete-algebraic approach to non-probabilistic programs of Back and von Wright [BvW99]. Unlike probabilistic Kleene algebra [MW05], this algebra contains both operators for expressing terminating *and* possibly non-terminating iterations. We consider the ability to express programs that are possibly non-terminating as important: in non-reactive programs a non-terminating loop is a classical programming error. Also, reactive programs—programs in which the behaviour over time is visible—may exhibit non-terminating behaviours.

One very important feature of this algebra is its simplicity. Like probabilistic Kleene algebra, the algebra has operators to represent sequential composition, choice and iterations, but it does not contain a probabilistic choice operator, or other probabilistic-program specific attributes. This decision reflects an important observation: many non-trivial transformation rules for probabilistic systems, such as the data refinement and separation and reduction rules we derive in Sect. 8, may in fact be specified and verified *without* having to reason directly about probabilistic choices or other probabilistic-program specific attributes. The generality of the algebra not only allows us to hide unnecessary details, but it allows us to use the algebra to capture similarities between different models—both probabilistic *and* non-probabilistic. This implies that results verified in the algebra are valid across a range of models, including

- Non-reactive non-probabilistic program models like the *isotone predicate transformers* which may be used to model programs with two forms of nondeterministic choice, angelic and demonic choice, in a total-correctness framework. Given a game-theoretic interpretation, demonic nondeterministic choice, $\sqcap$, represents a choice which cannot be controlled by someone observing the execution of a program, and which, if possible, will be made to her disadvantage, while an angelic choice, $\sqcup$, can be favourably influenced by the same observer in

order to achieve a desired outcome. These two forms of choice together may be used for modelling two-player games and contracts [BvW98].

- Non-reactive probabilistic program models like the *nondeterministic expectation transformers*, or the *dually nondeterministic expectation transformers* [MM01a, MH08b]. The first of these models, the nondeterministic expectation transformers, may be used to express probabilistic programs with a total-correctness semantics in which discrete probabilistic choice and demonic nondeterministic choice are expressible, while the second model, the dually nondeterministic expectation transformers, generalises the first by additionally allowing the expression of angelic choice.

Reactive probabilistic program models, including that described by Meinicke and Solin in [MS08], seem to also be likely models for the algebra.

We also introduce guards and assertions into the refinement algebra. Guards form a Boolean subalgebra of the carrier set and can be used when modelling 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 or terminates, respectively. The last-mentioned operators are similar to the domain operator of relational algebra and thus familiar to several people with a background in computer science or discrete mathematics.

The contents of the paper are organised as follows. First, the general refinement algebras that are used in the paper are identified and their suitability for probabilistic program models is explained. Guards and assertions are then introduced in Sect. 4. In Sect. 5, enabledness and termination operators are defined in the algebra and we discuss how they may be interpreted for probabilistic program models. A discussion of the limitations of using such a simple algebra to reason about probabilistic programs appears in Sect. 6, and basic properties which may be verified in the algebra are summarised in Sect. 7. We then explore and demonstrate the capabilities of the algebra in Sect. 8 by using it to reason about transformation theorems for probabilistic action systems. First we demonstrate that some fundamental action system transformation and data refinement theorems from the model-based work of Meinicke and Hayes [MH06, MH08b] may be given simple proofs in the abstract algebra. By lifting these earlier model-specific results into the abstract algebra, we demonstrate their validity across a wider range of models, as well as demonstrating the ability of the algebra to verify practical and important results. We then use the algebra to derive new separation and reduction theorems for probabilistic action systems. We conclude in Sect. 9. Details of two probabilistic program models, expectation-transformer models [MMS96, MM05, MM01a], that are used throughout the text to motivate the algebra are provided in Appendix A: Appendix A.1 briefly outlines the expectation-transformer theory of Morgan and McIver [MM01a], and in Appendix A.2 we justify the soundness of our axiomatisation with respect to the models from Appendix A.1. In Appendix A.3 we verify some other properties which are used to support the discussions in the text.

2. A very general refinement algebra

We use the term *refinement algebra* to refer to the set of abstract algebras closely related to Kleene algebra that are suitable for reasoning about programs in a total-correctness framework. The most studied of these algebras is von Wright's *demonic refinement algebra* (dRA) [vW02, vW04], a variant of Kleene algebra which is sound with respect to program models, like the infinitely conjunctive predicate transformers, in which only one form of nondeterministic choice is expressible. As noted by von Wright [vW04], the demonic refinement algebra is not suitable for program models like the isotone predicate transformers in which both demonic and angelic choices are present. As a result, he proposed a generalised algebra, the *general refinement algebra* (gRA), which is suitable for such a model. This algebra, which is also related to the algebras presented in [MCM06, TF06, MÖ4], is relatively new and unexplored. The axioms of the demonic refinement algebra are also not sound with respect to program models, like those used in [MH08b], in which discrete probabilistic choice as well as either one or two forms of nondeterministic choice coexist. However, we observe here that the general refinement algebra axioms are, and that this algebra is a very simple, but powerful tool for probabilistic program reasoning.

In this section we outline von Wright's general refinement algebra, and we discuss its suitability for probabilistic programs. We also investigate and develop further the basic theory of the algebra. The two main motivating probabilistic program models we refer to are the *nondeterministic* and the *dually nondeterministic* expectation transformers [MMS96, MM05, MM01a] which were investigated in the concrete-algebraic work of Meinicke and Hayes [MH08b]. Details of these models and soundness arguments may be found in Appendixes A.1 and A.2.

2.0.1. Axiomatisation

The general refinement 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 modelled 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).[1]

Definition 2.1 A *general refinement algebra* (gRA) 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):[2]

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

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 (8) that

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

[1] The one not familiar with this form of intuition can consult Back and von Wright [BvW98].

[2] Note that a typographical error appears in [vW04]: the distributivity axioms (10) and (11) are mistakenly written as $x(y \sqcap z) = xy \sqcap xz$ and $(x \sqcap y)z \sqsubseteq xz \sqcap yz$.

hold for any x [vW02], 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 general refinement algebra is sound with respect to both the set of nondeterministic, and dually nondeterministic expectation transformers over a possibly infinite state space. (See Appendixes A.1 and A.2.)

2.0.2. Discussion

Let us look a bit closer at some of the axioms. Like demonic refinement algebra, most of those not pertaining to the strong iteration operator should be familiar from Kleene algebra: in fact two of the Kleene algebra axioms are absent and one has been modified. The annihilation axiom $x\top = \top$ is absent since it is not suitable for models, like the expectation-transformer models used in [MH08b], in which non-terminating behaviour cannot be modified by executing subsequent commands—including $\top$. For the same reason it is also excluded from other algebras including the demonic refinement algebra [vW02] and Möller's Lazy Kleene algebra [MÖ04]. The remaining two axioms from Kleene algebra (and demonic refinement algebra) which have been, respectively, modified and elided are right distributivity,

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

and the induction axiom

$$x \sqsubseteq xy \sqcap z \Rightarrow x \sqsubseteq zy^*. \tag{19}$$

As noted by Meinicke and Hayes [MH06], these properties do not hold for programs which exhibit branching behaviour, and so right-distributivity is suitably weakened to right sub-distributivity (axiom (10)) and the induction axiom is elided. Although the induction axiom is elided in the general algebra, a weakening of this axiom has been shown to be valid for general refinement algebras that satisfy additional properties.

Any general refinement algebra that is also a complete lattice[3] (induced by the ordering $\sqsubseteq$), and for which each element x of the carrier set is semi-cocontinuous[4] has the following induction property for all x, y and z in the carrier set:

$$x \sqsubseteq x(y \sqcap 1) \sqcap z \Rightarrow x \sqsubseteq zy^*. \tag{20}$$

We thus refer to a general refinement algebra which takes (20) as an additional axiom as a *semi-cocontinuous general refinement algebra* (scc. gRA).[5] The axiom (20) is independent of the axioms of general refinement algebra, just as (19) is also independent [Koz90]: in fact, the same example used by Kozen in [Koz90], may also be used to prove independence in our case.

For finite state spaces, both the set of nondeterministic and dually nondeterministic expectation transformers are complete lattices whose elements are semi-cocontinuous [MH08b]. Consequently, as shown by Meinicke and Hayes [MH08b], they also satisfy unfolding axiom (20). On the other hand, for infinite state spaces, semi-continuity is lost in both models and axiom (20) fails to hold [Mei08].

In gRA, we have that axiom (13) is equivalent to

$$x \sqsubseteq yx \Rightarrow x \sqsubseteq y^*x, \tag{21}$$

and (20) is equivalent to

$$x \sqsubseteq x(y \sqcap 1) \Rightarrow x \sqsubseteq xy^*. \tag{22}$$

This may be verified in the same way as the equivalence of axiom (13) and (21), and of (19) and

$$x \sqsubseteq xz \Rightarrow x \sqsubseteq xz^*, \tag{23}$$

in Kleene algebra [Koz94].

---

[3] That is, each subset of the carrier set has a least upper bound and a greatest lower bound [DP90].

[4] An element x of a complete lattice is *semi-cocontinuous* if for all non-empty codirected subsets of the carrier set Y, $x;\ (\sqcap y \in Y \bullet y) = (\sqcap y \in Y \bullet x;\ y)$, where a set Y is codirected iff $(\forall y_1, y_2 \in Y \bullet (\exists\, y_3 \in Y \bullet y_3 \sqsubseteq y_1 \wedge y_3 \sqsubseteq y_2))$, and the greatest lower bound of a set of elements Y is written as $(\sqcap y \in Y \bullet y)$.

[5] We have earlier referred to this structure as a probabilistic demonic refinement algebra, but we find scc. gRA more apt.

Property (22) is taken as an axiom in the probabilistic Kleene algebra [MCM06] instead of (20). As mentioned above, it is not always valid for probabilistic models, in particular infinite state space models—this motivates its absence in gRA.

In Kleene algebra (and dRA), the induction axiom (19) has a "matching" unfolding property,

$$x^* = 1 \sqcap x^* x, \tag{24}$$

which is derivable from the axioms, excluding (19). In general refinement algebra, this property is not derivable, however it has another counterpart. An analogous property also holds true of strong iteration, despite the fact that it only has one induction rule.[6]

Proposition 2.2 For any x in the carrier set of a gRA, the equations

$$x^* = x^*(x \sqcap 1) \sqcap 1 \tag{25}$$
$$x^\omega = x^\omega(x \sqcap 1) \sqcap 1 \tag{26}$$

hold.[7]

Proof. The first claim is proved by

$$\begin{array}{ll} & x^*(x \sqcap 1) \sqcap 1 \\ \sqsubseteq & \{\text{isotony}\} \\ & x^* 1 \sqcap 1 \\ \sqsubseteq & \{1 \text{ is unit (8)}, \text{isotony}\} \\ & x^* \end{array}$$

and

$$\begin{array}{ll} & x^* \sqsubseteq x^*(x \sqcap 1) \sqcap 1 \\ \Leftarrow & \{x^* \sqsubseteq 1 \text{ and isotony}\} \\ & x^* \sqsubseteq x^*(x \sqcap 1) \\ \Leftarrow & \{\text{induction (13)}\} \\ & x^* \sqsubseteq xx^* \sqcap x \sqcap 1 \\ \Leftrightarrow & \{xx^* \sqsubseteq x\} \\ & x^* \sqsubseteq xx^* \sqcap 1 \\ \Leftrightarrow & \{\text{unfolding (12)}\} \\ & \mathsf{true}. \end{array}$$

The reverse refinement of the second claim is proved exactly as the first, and

$$\begin{array}{ll} & x^\omega \sqsubseteq x^\omega(x \sqcap 1) \sqcap 1 \\ \Leftarrow & \{x^\omega \sqsubseteq 1 \text{ and isotony}\} \\ & x^\omega \sqsubseteq x^\omega(x \sqcap 1) \\ \Leftarrow & \{\text{induction (15)}\} \\ & xx^\omega(x \sqcap 1) \sqcap 1 \sqsubseteq x^\omega(x \sqcap 1) \\ \Leftrightarrow & \{\text{unfolding (14)}, \text{distributivity (11)}\} \\ & xx^\omega(x \sqcap 1) \sqcap 1 \sqsubseteq xx^\omega(x \sqcap 1) \sqcap x \sqcap 1 \\ \Leftrightarrow & \{\text{commutativity of choice (4)}, \text{basic poset property}\} \\ & xx^\omega(x \sqcap 1) \sqcap 1 \sqsubseteq x \\ \Leftarrow & \{\text{basic poset property}\} \\ & xx^\omega(x \sqcap 1) \sqsubseteq x \\ \Leftarrow & \{\text{isotony}\} \\ & x^\omega(x \sqcap 1) \sqsubseteq 1 \\ \Leftarrow & \{x^\omega \sqsubseteq 1, \text{isotony}, 1 \text{ is unit (8)}\} \\ & \mathsf{true} \end{array}$$

gives the other direction. □

[6] This also means that in monodic tree Kleene algebra [TF06] the axiom $1 + x^*(x + 1) \leq x^*$ could actually be elided.

[7] Note that the more general properties $zx^* = zx^*(x \sqcap 1) \sqcap z$ and $zx^\omega = zx^\omega(x \sqcap 1) \sqcap z$ also hold.

As in the demonic refinement algebra [vW02], we include unfolding (14) and induction (15) axioms for the strong iteration operator. Unlike the demonic refinement algebra and lazy omega algebra [MÖ04], general refinement 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 an infinite number of iterations: this property is not satisfied by probabilistic—nor angelic—programs [MH06].

Although the primary focus of this paper is performing total correctness reasoning, we do not exclude discussions of the weak-iteration operator, since it may (in some circumstances) be applied when it is reasonable to assume that an iteration is terminating (see [MCM06, MH06]). In Sect. 8.1 we show how weak iteration, via the new scc. gRA axiom (20), plays an important role in the derivation of a data refinement rule for probabilistic action systems.

3. Healthiness conditions

In this section we introduce "healthiness conditions" to 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.

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

$$x(y \sqcap z) = xy \sqcap xz \tag{27}$$

for any y and z in the carrier set. As mentioned in the previous section, conjunctivity is not satisfied by all probabilistic programs. It is, however, satisfied by a large subset of these: the programs which do not include probabilistic (nor angelic) choices. When it is reasonable to take conjunctivity 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

$$(\forall n \bullet x(yn \sqcap u) \sqsubseteq zxn \sqcap vx) \Rightarrow xy^\omega u \sqsubseteq z^\omega vx \tag{28}$$

holds for any y, z, u and v in the carrier set (the bound variable n also ranges over the carrier set). 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 (see Appendix A). To define continuity in the familiar fashion, we would need an operator for expressing unbounded angelic choice, and this is not readily available in the present framework. Continuity is required in order to prove some useful commutativity rules for iterations.

4. Guards and assertions

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 this is the case, otherwise behave like magic. Guards must be introduced slightly differently in general refinement algebra than in demonic refinement algebra, since not every element is conjunctive but we still want guards to satisfy conjunctivity. Hence, along the lines of Solin [Sol06], an element g of the carrier set that

- is conjunctive and
- has a conjunctive complement $\bar{g}$ satisfying $g\bar{g} = \bar{g}g = \top$ 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 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 [vW02].

Every guard is defined to have a corresponding *assertion*

$$g^\circ = \bar{g}\bot \sqcap 1 \tag{29}$$

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

$$g^\circ = \bar{g}\bot \sqcap g \tag{30}$$

can be shown equivalent to (29). It is easy to show that

$$g_1{}^\circ \sqsubseteq 1 \sqsubseteq g_2 \tag{31}$$

holds for any assertion $g_1{}^\circ$ and any guard g_2 [vW02]. The properties

$$gg^\circ = g \quad \text{and} \quad g^\circ g = g^\circ \tag{32}$$

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

As already mentioned, gRA specifically requires that guards are conjunctive, which differs from dRA in which *all* elements by axiomatisation are conjunctive. On the other hand, the derivation

$$\begin{aligned}
& g^\circ(x \sqcap y) \\
= \; & \{\text{assertion definition (29)}\} \\
& (\bar{g}\bot \sqcap 1)(x \sqcap y) \\
= \; & \{\text{distributivity (11)}\} \\
& \bar{g}\bot(x \sqcap y) \sqcap x \sqcap y \\
= \; & \{\bot \text{ is left-annihilating (17), idempotence (6) and commutativity (4)}\} \\
& \bar{g}\bot \sqcap x \sqcap \bar{g}\bot \sqcap y \\
= \; & \{\bot \text{ is left-annihilating (17)}\} \\
& \bar{g}\bot x \sqcap x \sqcap \bar{g}\bot y \sqcap y \\
= \; & \{\text{distributivity (11) and assertion definition (29)}\} \\
& g^\circ x \sqcap g^\circ y
\end{aligned}$$

proves that the conjunctivity of assertions follows from the conjunctivity of guards.

5. Enabledness and termination

Solin and von Wright have extended the demonic refinement algebra with operators for determining when elements of the carrier set are enabled or terminating [SvW06], in a similar way to which Desharnais, Möller and Struth added a domain operator to Kleene algebra [DMS06]. Here we include these operators in gRA and describe how they may be given an interpretation for probabilistic programs.

5.1. Enabledness

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

$$\epsilon x x = x, \tag{33}$$

$$g \sqsubseteq \epsilon(gx) \text{ and} \tag{34}$$

$$\epsilon(xy) = \epsilon(x\epsilon y). \tag{35}$$

We will call a gRA with an enabledness operator a gRAe.

The probabilistic program intuition of ϵx is that it returns a guard that skips in those states from which x is not miraculous with probability one. That is to say, ϵx checks whether the program is certainly disabled or not. The first axiom, (33), thus says that a program is equal to the same program preceded by a guard that checks if it is not certainly disabled: if the program is not certainly disabled, the guard skips and then executes the program, if the program is certainly disabled (that is, it is will perform a miracle with probability one), the guard will not hold and thus the whole program will do magic. The other axioms can be interpreted similarly.

The definition of enabledness is the same as that used to describe the domain operator in Kleene algebra with domain [DMS06], and the enabledness operator in the demonic refinement algebra [SvW06]. A stronger enabledness operator satisfying also the axiom

$$\epsilon x \bot = x \bot, \tag{36}$$

has also been considered in demonic refinement algebra [SvW06] and other places [Ehm03]. Axiom (36) does not hold in our probabilistic context (cf. Appendix A, Example A.2). Let us consider why this is the case. Property (36) states that the least element $\bot$ is able to right-annihilate any part of an element x which is enabled. This is not valid in our probabilistic interpretation of the enabledness operator, since an "enabled" program may still be miraculous with some probability greater than zero, and $\bot$ may not annihilate this miraculous part of the program. (The interested reader may wish to refer to Sect. 8.1 for an example of how this algebraic difference affects a practical program transformation rule.)

The properties

$$\epsilon(x \sqcap y) = \epsilon x \sqcap \epsilon y \text{ and} \tag{37}$$

$$x \sqsubseteq y \Rightarrow \epsilon x \sqsubseteq \epsilon y \tag{38}$$

can be shown to hold by similar proofs as for the domain operator in [DMS06], taking into consideration that guards are conjunctive.

Given the assumption that the state space is finite, enabledness operators which satisfy axioms (33–35) may be defined in both our motivating expectation-transformer models. Interestingly however, for the case when the state space is infinite, satisfaction of axiom (35) is lost for the class of dually nondeterministic transformers (cf. Appendix A, Example A.1).

We show how the enabledness operator may be used to express and reason about probabilistic action systems in Sect. 8.

5.2. Termination

We define the *termination operator* τ to be a unary operator that maps a carrier-set element of gRA to an assertion that satisfies

$$x = \tau x x, \tag{39}$$

$$\tau(g^\circ x) \sqsubseteq g^\circ, \tag{40}$$

$$\tau(x \tau y) = \tau(xy) \text{ and} \tag{41}$$

$$\tau(x \sqcap y) = \tau x \sqcap \tau y. \tag{42}$$

We call a gRA with termination gRAt, and a gRA with enabledness and termination gRAet.

The intuition behind the probabilistic interpretation of the termination operator is similar to that for enabledness. For an element x, τx denotes an assertion which checks whether a program *does not certainly* abort—that is, it has some non-zero chance of not aborting. Let us look at the first property. 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.

We use the same definition of termination as is taken in demonic refinement algebra [SvW06]. Similar to the enabledness operator, the additional axiom

$$\tau x \top = x \top \tag{43}$$

has also been included in definitions of termination in demonic refinement algebra [SvW06]. This axiom is so strong that the other termination axioms may indeed be derived from it [SvW06]. Property (43) is not valid in our motivating models for a similar reason to why enabledness property (36) is not suitable.

Given the assumption that the state space is finite, termination operators which satisfy axioms (39–42) may be defined in both our motivating expectation-transformer models. For the case when the state space is infinite, satisfaction of axiom (41) is lost (cf. Appendix A, Example A.3).

Property

$$x^\omega \tau x = x^\omega \tag{44}$$

may be verified in gRAt by

$$\begin{array}{ll} & x^\omega \sqsubseteq x^\omega \tau x \\ \Leftarrow & \{\text{induction (15)}\} \\ & x x^\omega \tau x \sqcap 1 \sqsubseteq x^\omega \tau x \\ \Leftrightarrow & \{\text{unfolding (14), left distributivity (11)}\} \\ & x x^\omega \tau x \sqcap 1 \sqsubseteq x x^\omega \tau x \sqcap \tau x \\ \Leftrightarrow & \{\text{termination axiom (39) and assertion property } g^\circ x \sqcap y = g^\circ x \sqcap g^\circ y\} \\ & x x^\omega \tau x \sqcap \tau x \sqsubseteq x x^\omega \tau x \sqcap \tau x \\ \Leftrightarrow & \mathsf{true}, \end{array}$$

and is used in the proof of a data refinement theorem in Sect. 8.

6. Limitations of the algebra

We have chosen to work with the general refinement algebra because of its simplicity and generality. However, with generality comes some limitations. In this section we attempt to give an brief overview of some of these constraints.

First, it is evident that we cannot perform explicit reasoning about probabilistic choices. Although, as identified in the introduction, many fundamental transformation theorems only require implicit reasoning about this operator. Further investigations into probabilistic extensions to gRA may be found [MH08a]. The work here is foundational to such extensions.

Another more subtle point is that since the algebra allows us to capture similarities between probabilistic program models that both may, and may not, contain angelic choices, algebraic properties that are specific to the more restrictive model can certainly *not* be verified in the algebra. So, what are some of these important differences that can be expressed in the algebra, and what impact do they have on our ability to reason about probabilistic program transformations?

The algebraic differences between our two motivating probabilistic models—which can be expressed without the use of the probabilistic choice operator—tend to be rather subtle. For instance, consider the following implication,

$$(\top = px\bar{q}) \Rightarrow (pxq = px), \tag{45}$$

where x and guards p and q are elements from the carrier set. Condition $\top = px\bar{q}$ describes a *total correctness assertion* which has equivalent formulations

$$(\bar{p}x \sqsubseteq x\bar{q}) \Leftrightarrow (\bar{p}x\bar{q} \sqsubseteq x\bar{q}) \Leftrightarrow (\bar{p}\bot \sqsubseteq x\bar{q})$$

and $pxq = px$ embodies what we refer to as a *weak correctness assertion*, which may be expressed equivalently as

$$(xq \sqsubseteq px) \Leftrightarrow (p^\circ xq^\circ = p^\circ x) \Leftrightarrow (p^\circ x \sqsubseteq xq^\circ)$$

in gRA.[8] Given a program interpretation, the total correctness assertion can be taken to mean that from an initial state in which p holds, program x can be guaranteed to terminate in a state satisfying q; while the weak correctness assertion states that from an initial state in which p holds, x may either fail to terminate *or* it must reach a state in which q holds.

In the nondeterministic expectation-transformer model property (45) holds, but in the dually nondeterministic model, it does not—see Appendix A.3. This means that (45) cannot be derived in gRA, and may be used as evidence that gRA is not complete for our non-angelic motivating model.

Another important difference between the angelic and non-angelic probabilistic models is that, for the non-angelic model, certain termination of a strong iteration statement x^ω, guarantees that x^ω can be equated with x^*. In the algebra this may be expressed by

$$(x^\omega \top = \top) \Rightarrow (x^\omega = x^*). \tag{46}$$

This property also does not hold in general in the angelic model [MH08b].

[8] These equivalences may be justified using the same arguments as those employed by von Wright in dRA [vW02].

Interestingly, these algebraic differences are rarely required to verify transformation rules for non-angelic probabilistic programs, and so we prefer to take them as assumptions where required, rather than to jeopardise the generality of the algebra—and therefore our results—by specialising it further. (For interest, we demonstrate a place where (45) may be useful in Sect. 8.1. Property (46) could, for example, be used in practice to simplify assumptions to the propositions that appear later in Sect. 8.2.3.) This incompleteness result does not undermine the usefulness of the algebra: the general refinement algebra provides a simple and clear way to explain key features of probabilistic programs and to reduce the verification of complex theorems down to simpler properties which may then, if necessary, be verified in a chosen model itself.

7. Basic algebraic properties

In this section we summarise some basic properties of the algebra which will be used in subsequent proofs. Many of these results have been employed by Meinicke and Hayes [MH06, MH08b] in a concrete-algebraic setting.

For any x and y, and guard g in the carrier set of a gRA

$$(x \sqcap y)^\omega = x^\omega (yx^\omega)^\omega, \tag{47}$$

$$x(yx)^\omega \sqsubseteq (xy)^\omega x, \tag{48}$$

$$x(yx)^\omega = (xy)^\omega x, \text{ provided } x \text{ is conjunctive,} \tag{49}$$

$$x^* = x^* x^*, \tag{50}$$

$$x^\omega = x^\omega x^\omega, \tag{51}$$

$$(x \sqcap y)^\omega = y^\omega (x \sqcap y)^\omega, \tag{52}$$

$$(x \sqcap y)^\omega = (x \sqcap y)^\omega y^\omega, \tag{53}$$

$$(gx = gxg) \Rightarrow gx^\omega = g(gx)^\omega = gx^\omega g \text{ and} \tag{54}$$

$$\overline{g}(gx)^\omega = \overline{g} \tag{55}$$

hold. Decomposition (47) and leapfrog property (48) have already been verified in gRA [vW04], and the others are simple to derive.

7.0.1. Commutativity properties

For any x and y in the carrier set of a gRA the following commutativity properties hold:

$$xy^* \sqsubseteq zx \Rightarrow xy^* \sqsubseteq z^* x, \tag{56}$$

$$xy \sqsubseteq zx \Rightarrow xy^\omega \sqsubseteq z^\omega x, \text{ provided } x \text{ is continuous, and} \tag{57}$$

$$yx \sqsubseteq zy \Rightarrow y^\omega x \sqsubseteq zy^\omega, \text{ provided } x \text{ is conjunctive.} \tag{58}$$

Assuming continuity of x we can for example see that (57) holds since, for any n,

$$\begin{aligned} & x(yn \sqcap 1) \\ \sqsubseteq\ & \{\text{right sub-distributivity (10), 1 is unit (8)}\} \\ & xyn \sqcap x \\ \sqsubseteq\ & \{\text{assumptions } xy \sqsubseteq zx, \text{ isotony, 1 is unit (8)}\} \\ & zxn \sqcap 1x. \end{aligned}$$

The commutativity property

$$y^*x \sqsubseteq x(z \sqcap 1) \Rightarrow y^*x \sqsubseteq xz^* \tag{59}$$

holds only if the new scc. gRA induction axiom (20) may be assumed. It may be verified as follows:

$$\begin{aligned}
& y^*x \sqsubseteq x(z \sqcap 1) \\
\Rightarrow\ & \{\text{isotony}\} \\
& y^*y^*x \sqsubseteq y^*x(z \sqcap 1) \\
\Leftrightarrow\ & \{\text{basic property } y^*y^* = y^* \text{ (50)}\} \\
& y^*x \sqsubseteq y^*x(z \sqcap 1) \\
\Leftrightarrow\ & \{y^*x(z \sqcap 1) \sqsubseteq x \text{ follows from unfolding (12), isotony, 1 is unit (8)}\} \\
& y^*x \sqsubseteq y^*x(z \sqcap 1) \sqcap x \\
\Rightarrow\ & \{\text{new weak iteration induction axiom (20)}\} \\
& y^*x \sqsubseteq xz^*.
\end{aligned}$$

Note that since (59) is a generalisation of (22), property (59) and (20) are actually equivalent in gRA.

7.0.2. Basic separation and reduction theorem

The derivation of more complex separation and reduction theorems for probabilistic action systems, requires us to understand the situations under which we can decompose an iteration $(x \sqcap y)^\omega$ to a simpler form $x^\omega y^\omega$, in which element x and y are iterated without interference from each other. As in probabilistic Kleene algebra [MW05], the equivalent theorem for weak iteration,

$$(x \sqcap 1)y^* \sqsubseteq y(x \sqcap 1) \Rightarrow x^*y^* = (x \sqcap y)^* \tag{60}$$

is derivable in scc. gRA. The corresponding theorem for strong iteration [MH08b] requires the use of control variables. Here we simplify the proof from [MH08b], replaying it in an abstract algebraic setting. First we explain how we reason about control variables in the abstract algebra, and then we present and verify the basic separation and reduction theorem for strong iteration.

Control variables in the abstract algebra. When reasoning about program transformations it is often necessary to introduce and reason about *control variables* (see for example [Koz97, PK00, vW02]). In our context, a control variable may be manipulated using three elements of the carrier set: a guard g, which checks to see if the control variable is set or unset, and elements n and m, which are used, respectively, to set and unset the control variable. Formally, we say that (g, n, m) are *control elements* if g is a guard, n and m are continuous and conjunctive and

$$n = ng, \tag{61}$$

$$m = m\bar{g} \text{ and} \tag{62}$$

$$m = mm. \tag{63}$$

Intuitively, the first two conditions describe the fact that n switches on the control variable, and m switches it off. The third equivalence then says that repeatedly switching off the variable has the same effect as performing the statement once.

Such control elements are to be used in the context of a program that is comprised of a number of elements x_i, for i in some finite set I, which do not refer directly to the control variable. We say that control elements (g, n, m) *do not interfere* with elements $\{i \in I \cdot x_i\}$ if for all $i \in I$

$$gx_i = gx_ig, \tag{64}$$

$$\bar{g}x_i = \bar{g}x_i\bar{g}, \tag{65}$$

$$x_in = nx_i \text{ and} \tag{66}$$

$$x_im = mx_i. \tag{67}$$

In the following theorems we implicitly make the extra assumption that given control statements (g, n, m), which do not interfere with the elements in x and y, $nxm = nym$ implies that $x = y$. This is reasonable for program models in which m and n manipulate variables which do not occur free in x and y.

The basic theorem. The following theorem is phrased in a similar way to (60), although—since we are reasoning about possibly infinite iterations—we must replace the occurrence of 1 in $(x \sqcap 1)$ with an element m, whose behaviour is disjoint from x. This is required in order to disallow y from possibly enabling a behaviour of x which is equivalent to 1. This extra constraint is necessary since infinite iterations of 1 are synonymous with $\bot$.

Proposition 7.1 *If x and y are elements of the carrier set, there exist control elements (g, n, m) that do not interfere with x* and *y, and either*

1. *$(x \sqcap m)$ is continuous and $(x \sqcap m)y \sqsubseteq y(x \sqcap m)$*, or
2. *$y^\omega = y^*$ and $(x \sqcap m)y^* \sqsubseteq y(x \sqcap m)$,*

then we have that $x^\omega y^\omega = (x \sqcap y)^\omega$.

Proof. We have that $(x \sqcap y)^\omega \sqsubseteq x^\omega y^\omega$ follows from rule $x^\omega = x^\omega x^\omega$ (51) and isotony. For the proof in the other direction we use the fact that the control statement assumptions together with (58), (57), (54) can be used to show that

$$y^\omega m = m y^\omega m, \tag{68}$$
$$g y^\omega = g y^\omega g, \tag{69}$$
$$g(x \sqcap y)^\omega = g(gx \sqcap y)^\omega \text{ and} \tag{70}$$
$$\bar{g} y^\omega = \bar{g}(gx \sqcap y)^\omega. \tag{71}$$

The proof proceeds as follows:

$x^\omega y^\omega \sqsubseteq (x \sqcap y)^\omega$
$\Leftrightarrow$ {assumptions on elements for manipulating control variables}
$nx^\omega y^\omega m \sqsubseteq ng(x \sqcap y)^\omega m$
$\Leftarrow$ {isotony}
$x^\omega y^\omega m \sqsubseteq g(x \sqcap y)^\omega m$
$\Leftarrow$ {induction (15)}
$xg(x \sqcap y)^\omega m \sqcap y^\omega m \sqsubseteq g(x \sqcap y)^\omega m$
$\Leftrightarrow$ {$(x \sqcap y)^\omega = (x \sqcap y)^\omega y^\omega$ (53), decomposition (47)}
$xg(x \sqcap y)^\omega m \sqcap y^\omega m \sqsubseteq g y^\omega (xy^\omega)^\omega y^\omega m$
$\Leftrightarrow$ {unfolding (14)}
$xg(x \sqcap y)^\omega m \sqcap y^\omega m \sqsubseteq g y^\omega (xy^\omega (xy^\omega)^\omega y^\omega m \sqcap y^\omega m)$
$\Leftrightarrow$ {decomposition (47) and $(x \sqcap y)^\omega = (x \sqcap y)^\omega y^\omega$ (53)}
$xg(x \sqcap y)^\omega m \sqcap y^\omega m \sqsubseteq g y^\omega (x(x \sqcap y)^\omega m \sqcap y^\omega m)$
$\Leftrightarrow$ {(68) and (69), conjunctivity of guards and control statement assumptions}
$xg(x \sqcap y)^\omega m \sqcap m y^\omega m \sqsubseteq g y^\omega (xg(x \sqcap y)^\omega m \sqcap m y^\omega m)$
$\Leftrightarrow$ {(70) and (71)}
$xg(gx \sqcap y)^\omega m \sqcap m(gx \sqcap y)^\omega m \sqsubseteq g y^\omega (xg(gx \sqcap y)^\omega m \sqcap m(gx \sqcap y)^\omega m)$
$\Leftrightarrow$ {left distributivity (11) and $(x \sqcap y)^\omega = y^\omega (x \sqcap y)^\omega$ (52)}
$(xg \sqcap m) y^\omega (gx \sqcap y)^\omega m \sqsubseteq g y^\omega (xg \sqcap m)(gx \sqcap y)^\omega m$
$\Leftarrow$ {isotony, guards refine 1 (31)}
$g(xg \sqcap m) y^\omega \sqsubseteq g y^\omega (xg \sqcap m)$
$\Leftrightarrow$ {(69), conjunctivity of guards, control statement assumptions}
$g(x \sqcap m) y^\omega \sqsubseteq g y^\omega (x \sqcap m)$.

Using commutativity properties (56) and (57), we then have that this last refinement holds if either the first or second set of hypotheses are assumed. □

8. Probabilistic action systems

This section comprises an application of the refinement algebra to probabilistic action systems. Action systems can be used for reasoning about parallel or distributed systems in which concurrent behaviour is modelled by interleaving atomic actions [BKS83]. Probabilistic action systems extend action systems to account also for probabilistic behaviour, in that the actions are allowed to be probabilistic programs [ST96]. An action system

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

is an iteration of a set of *actions* $x_1, \ldots, x_n$ 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 [SvW06, BvW99, MH06]:

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

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.

8.1. Program refinement

Two transformation rules that have been discussed in the literature on algebraic reasoning about program refinement are action-system leapfrog and decomposition. In this section we consider these rules from our current probabilistic abstract-algebraic perspective. We begin with the leapfrog rule.

Proposition 8.1 *For elements x and y in the carrier set of a* gRAe, *if*

$$x\overline{\epsilon(yx)} \sqsubseteq \overline{\epsilon(xy)}x, \tag{72}$$

then

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

holds.

This can be verified in the abstract algebra by the action system definition, isotony, the assumption, and leapfrog (48).

In dRAe, condition (72) may be shown to always hold [SvW06], but this does not hold true in gRAe (see [MH08b] for a counter example). Condition (72) can be proved in gRAe just like in dRAe [SvW06] if conjunctivity (27) of x is assumed. However, it is not necessary to assume conjunctivity: it is enough to assume that, for all guards p and q, x satisfies

$$(\top = px\overline{q}) \Rightarrow (pxq = px),$$

i.e. (45), which is valid in our non-angelic motivating model. Indeed, the proposition then holds since with the assumption the proof from [SvW06] can be followed in detail.

We now turn to the decomposition of action systems.

Proposition 8.2 *For elements x and y in the carrier set of a* gRAe, *if $\epsilon x \epsilon y = \top$ and $\epsilon x = \epsilon(x\ \mathsf{do}\ y\ \mathsf{od})$, then*

$$\mathsf{do}\ x \sqcap y\ \mathsf{od} = \mathsf{do}\ y\ \mathsf{od};\ \mathsf{do}\ x;\ \mathsf{do}\ y\ \mathsf{od}\ \ \mathsf{od}$$

holds.

This may be proved just like in [SvW06]. *The condition $\epsilon x = \epsilon(x\ \mathsf{do}\ y\ \mathsf{od})$ is true if $\epsilon(\mathsf{do}\ y\ \mathsf{od}) = 1$, as shown by*

$$\begin{aligned}
&\epsilon(x\ \mathsf{do}\ y\ \mathsf{od})\\
=\ &\{\text{enabledness axiom (35)}\}\\
&\epsilon(x\epsilon(\mathsf{do}\ y\ \mathsf{od}))\\
=\ &\{\text{assumption } \epsilon(\mathsf{do}\ y\ \mathsf{od}) = 1\}\\
&\epsilon(x1)\\
=\ &\{1 \text{ is unit (axiom 8)}\}\\
&\epsilon x.
\end{aligned}$$

If the property

$$\epsilon x \bot = x \bot,$$

i.e. the enabledness axiom that we exclude from out definition (36), is able to be assumed for y, then the condition $\epsilon(\mathsf{do}\ y\ \mathsf{od}) = 1$ can always be shown to hold [SvW06]. Interestingly, $\epsilon(\mathsf{do}\ y\ \mathsf{od}) = 1$ may *not* be derived using only the gRA axioms and the restricted definition of enabledness. Consider the following example from our motivating model:

$$\begin{aligned}
&\mathsf{do}\ (\mathsf{skip}\ {}_{\frac{1}{2}}\!\oplus\ \mathsf{magic})\ \mathsf{od}\\
=\ &(\mathsf{skip}\ {}_{\frac{1}{2}}\!\oplus\ \mathsf{magic})^{\omega};\ \mathsf{magic}\\
=\ &\mathsf{magic}
\end{aligned}$$

The action body is always enabled and so it must always continue execution. Each time the action body skip $\frac{1}{2}\oplus$ magic executes it has a non-zero probability of performing magic, and so if it iterates forever, it eventually executes magic with probability one.

8.2. Data refinement

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

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

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.

8.2.1. Downward simulation

Probabilistic action systems 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 first establish a general commutativity property.

Proposition 8.3 *For any x, y and z in the carrier set and g_1 and g_2 in the guard set of a* gRA *we have that*

$$(g_1 y)^\omega \overline{g_1} x \sqsubseteq x(g_2 z)^\omega \overline{g_2} \tag{73}$$

provided

$$g_1{}^\circ yx \sqsubseteq x g_2{}^\circ z \quad \textit{and} \tag{74}$$
$$x\overline{g_2} \sqsubseteq \overline{g_1}x. \tag{75}$$

Proof. This property was first proved in the concrete expectation-transformer algebra by Meinicke and Hayes [MH06]. We here show how their proof can be given as a smooth and transparent derivation in gRA: first, we have that

$$\begin{array}{ll} & (g_1 y)^\omega \overline{g_1} x \sqsubseteq x(g_2 z)^\omega \overline{g_2} \\ \Leftarrow & \{\text{induction (15)}\} \\ & g_1 yx(g_2 z)^\omega \overline{g_2} \sqcap \overline{g_1} x \sqsubseteq x(g_2 z)^\omega \overline{g_2} \end{array}$$

holds.

That the antecedent follows from the assumptions is settled by

$$
\begin{array}{ll}
 & g_1 y x (g_2 z)^\omega \overline{g_2} \sqcap \overline{g_1} x \\
= & \{(32)\} \\
 & g_1 g_1{}^\circ y x (g_2 z)^\omega \overline{g_2} \sqcap \overline{g_1} x \\
\sqsubseteq & \{\text{assumption (74), isotony}\} \\
 & g_1 x g_2{}^\circ z (g_2 z)^\omega \overline{g_2} \sqcap \overline{g_1} x \\
= & \{\text{assertion definition (30)}\} \\
 & g_1 x (\overline{g_2}\bot \sqcap g_2) z (g_2 z)^\omega \overline{g_2} \sqcap \overline{g_1} x \\
= & \{\text{distributivity (11) and } \bot \text{ is left-annihilating (17)}\} \\
 & g_1 x (\overline{g_2}\bot \sqcap g_2 z (g_2 z)^\omega \overline{g_2}) \sqcap \overline{g_1} x \\
\sqsubseteq & \{\bot \sqsubseteq 1, \text{commutativity (4)}\} \\
 & g_1 x (g_2 z (g_2 z)^\omega \overline{g_2} \sqcap \overline{g_2}) \sqcap \overline{g_1} x \\
= & \{\text{distributivity (11)}\} \\
 & g_1 x (g_2 z (g_2 z)^\omega \sqcap 1) \overline{g_2} \sqcap \overline{g_1} x \\
= & \{\text{folding (14)}\} \\
 & g_1 x (g_2 z)^\omega \overline{g_2} \sqcap \overline{g_1} x \\
\sqsubseteq & \{1 \text{ is unit (8), guards refine 1 (31)}\} \\
 & g_1 x (g_2 z)^\omega \overline{g_2} \sqcap \overline{g_1} x \overline{g_2} \\
= & \{\text{guards form a Boolean algebra, axioms (9) and (5)}\} \\
 & g_1 x (g_2 z)^\omega \overline{g_2} \sqcap \overline{g_1} x (\overline{g_2} g_2 z (g_2 z)^\omega \sqcap \overline{g_2}) \\
= & \{\text{guards are conjunctive}\} \\
 & g_1 x (g_2 z)^\omega \overline{g_2} \sqcap \overline{g_1} x \overline{g_2} (g_2 z (g_2 z)^\omega \sqcap 1) \\
\sqsubseteq & \{\text{folding (14), 1 is unit (8), guards refine 1 (31)}\} \\
 & g_1 x (g_2 z)^\omega \overline{g_2} \sqcap \overline{g_1} x \overline{g_2} (g_2 z)^\omega \overline{g_2} \\
\sqsubseteq & \{\text{assumption (75), guards form a Boolean algebra}\} \\
 & g_1 x (g_2 z)^\omega \overline{g_2} \sqcap \overline{g_1} x (g_2 z)^\omega \overline{g_2} \\
= & \{\text{distributivity (11)}\} \\
 & (g_1 \sqcap \overline{g_1}) x (g_2 z)^\omega \overline{g_2} \\
= & \{\text{definition of guards}\} \\
 & x (g_2 z)^\omega \overline{g_2}.
\end{array}
$$

□

The data refinement rule then follows in gRAe by setting g_1 to be ϵy, setting g_2 to be ϵz, and taking the first enabledness axiom (33) into account.

8.2.2. 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 in gRAet assuming that x is continuous, and that

$$
\begin{array}{r}
xy \sqsubseteq zx \text{ and} \\
x(\tau y)\overline{\epsilon y} \sqsubseteq \overline{\epsilon z} x
\end{array}
$$

hold, that is, assuming that y is upwards data refined by z through x, and assuming that $(\tau y)\overline{\epsilon y}$ 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:

$$
\begin{array}{ll}
 & x;\ \mathsf{do}\ y\ \mathsf{od} \\
= & \{\text{action system encoding and Property (44)}\} \\
 & x y^\omega (\tau y)\overline{\epsilon y} \\
\sqsubseteq & \{\text{assumptions } x \text{ continuous, } xy \sqsubseteq zx \text{ and (57)}\} \\
 & z^\omega x (\tau y)\overline{\epsilon y} \\
\sqsubseteq & \{\text{assumption } x(\tau y)\overline{\epsilon y} \sqsubseteq \overline{\epsilon z} x \text{ and action system definition}\} \\
 & \mathsf{do}\ z\ \mathsf{od};\ x.
\end{array}
$$

8.2.3. *General simulation*

Unlike both the upward and downward simulation rules that have been presented, the following simulation rules 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 these rules requires reasoning about both weak and strong iterations. For the downward simulation rule, the finite-iteration assumption allows us to use the weak iteration axiom (20) from scc. gRA, for which there is no counterpart for strong iteration. Axiom (20) is actually *required* to prove this rule, and so it is valid in scc. gRAe, but not gRAe.

The general downward simulation rule (from [MH08b]) 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

$$ {y_\natural}^* x \sqsubseteq x(z_\natural \sqcap 1), \tag{76}$$
$$ (\epsilon y)^\circ y_\sharp {y_\natural}^* x \sqsubseteq x(\epsilon z)^\circ z_\sharp \text{ and} \tag{77}$$
$$ x\overline{\epsilon z} \sqsubseteq \overline{\epsilon y}x. \tag{78}$$

We refer to $y_\natural$ and $z_\natural$ as the stuttering actions, and $y_\sharp$ and $z_\sharp$ as the nonstuttering actions. The proof of Meinicke and Hayes [MH08b] may then be expressed as a derivation in gRAe.

It is sufficient to show that the conditions

$$ {y_\natural}^* x \sqsubseteq x {z_\natural}^*, \tag{79}$$
$$ (\epsilon y)^\circ y_\sharp {y_\natural}^* x \sqsubseteq x(\epsilon z)^\circ z_\sharp {z_\natural}^* \text{ and} \tag{80}$$
$$ x\overline{\epsilon z} \sqsubseteq \overline{\epsilon y}x \tag{81}$$

hold, since then we can derive

$$\begin{aligned}
&\mathsf{do}\ y\ \mathsf{od};\ x \\
&= \{\text{action system encoding and assumption } y = y_\natural \sqcap y_\sharp\} \\
&\quad (y_\natural \sqcap y_\sharp)^\omega \overline{\epsilon y} x \\
&\sqsubseteq \{\text{decomposition (47) and guards refine 1 (31)}\} \\
&\quad y_\natural^\omega (\epsilon y y_\sharp y_\natural^\omega)^\omega \overline{\epsilon y} x \\
&\sqsubseteq \{\text{assumptions (80), (81) and } y_\natural^\omega = y_\natural^*, \text{ (73)}\} \\
&\quad y_\natural^\omega x (\epsilon z z_\sharp z_\natural^\omega)^\omega \overline{\epsilon z} \\
&= \{\text{assumption } y_\natural^\omega = y_\natural^*\} \\
&\quad y_\natural^* x (\epsilon z z_\sharp z_\natural^\omega)^\omega \overline{\epsilon z} \\
&\sqsubseteq \{\text{assumption (79)}\} \\
&\quad x z_\natural^* (\epsilon z z_\sharp z_\natural^\omega)^\omega \overline{\epsilon z} \\
&= \{\text{assumption } z_\natural^\omega = z_\natural^*\} \\
&\quad x z_\natural^\omega (\epsilon z z_\sharp z_\natural^\omega)^\omega \overline{\epsilon z} \\
&\sqsubseteq \{\text{assumption } z = z_\natural \sqcap z_\sharp, \text{ isotony}\} \\
&\quad x z_\natural^\omega (\epsilon z_\natural z_\sharp z_\natural^\omega)^\omega \overline{\epsilon z} \\
&= \{\text{axiom (33)}\} \\
&\quad x z_\natural^\omega (z_\sharp z_\natural^\omega)^\omega \overline{\epsilon z} \\
&= \{\text{decomposition (47), assumption } z = z_\natural \sqcap z_\sharp, \text{ encoding}\} \\
&\quad x;\ \mathsf{do}\ z\ \mathsf{od}.
\end{aligned}$$

Condition (79) may be shown to hold given (76) and (59).

Assuming (79) and (77), (80) can be shown to hold by

$$
\begin{aligned}
& (\epsilon y)^\circ y_\sharp {y_\natural}^* x \\
= & \{(50)\} \\
& (\epsilon y)^\circ y_\sharp {y_\natural}^* {y_\natural}^* x \\
\sqsubseteq & \{\text{assumption (79)}\} \\
& (\epsilon y)^\circ y_\sharp {y_\natural}^* x {z_\natural}^* \\
\sqsubseteq & \{\text{assumption (77)}\} \\
& x(\epsilon z)^\circ z_\sharp {z_\natural}^*.
\end{aligned}
$$

Conditions (81) and (78) are equal.

Similarly it is possible to verify the following general upward simulation property. We have that in gRAet

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

holds if x is continuous, $y = y_\natural \sqcap y_\sharp$, $z = z_\natural \sqcap z_\sharp$, ${y_\natural}^\omega = {y_\natural}^*$, ${z_\natural}^\omega = {z_\natural}^*$, and conditions

$$x{y_\natural}^* \sqsubseteq z_\natural x, \tag{82}$$

$$x y_\sharp {y_\natural}^* \sqsubseteq z_\sharp x, \text{ and} \tag{83}$$

$$x(\tau y)\overline{\epsilon y} \sqsubseteq \overline{\epsilon z} x, \tag{84}$$

hold.

8.3. Atomicity refinement

Separation and reduction theorems, or *atomicity refinement* theorems may be used to simplify the development and analysis of concurrent or distributed systems by describing the conditions under which it is safe to assume that certain sequences of operations are executed without interference from other actions.

In this section we derive a new version of Back's atomicity refinement theorem [Bac89, BvW99], which is valid for probabilistic action systems.

First we present two new general purpose theorems, and then we show how they may be applied to verify more specific atomicity refinement theorems for probabilistic action systems. The first theorem is presented in a similar way to the non-probabilistic generalisation of Back's atomicity refinement theorem which appears in [Coh00].

8.3.1. General purpose theorems

The first theorem we present may be used to describe when an iteration

$$\bar{p}(py \sqcap \bar{p}y \sqcap \bar{p}l \sqcap \bar{p}r\bar{p})^\omega \bar{p}$$

in which action y is not interrupted by actions l and r when it is in its critical section p, may be replaced by an iteration

$$\bar{p}(y \sqcap l \sqcap r)^\omega \bar{p}$$

in which executions of y when it is in its critical section, p, may be interleaved with the other actions l and r.[9]

The first two assumptions in this theorem state that action l cannot enable p (85), and that r cannot disable it (86). Commutativity assumptions (87–91) then specify that l must be able to commute to the left over the other actions, and that r must be able to commute to the right over y when it is in its critical section. The additional constraints (91) and (92) are also needed in order to verify that l and r can be shifted to the left and right, respectively, when y is in its critical section.

The main difference to the non-probabilistic versions of this theorem [BvW99, Coh00] is that we do not make conjunctivity assumptions on the actions, so as not to exclude the possibility that these actions may include probabilistic choices. As a result, the commutativity assumptions are expressed in a way which is compatible with the basic separation and reduction Proposition 7.1.

[9] Note that we could, in the spirit of Cohen's work [Coh00], attempt to generalise this theorem further by allowing the iteration to start and finish in its critical section.

Proposition 8.4 *Using assumptions that there exist control elements* (n, m, g) *that are independent of elements* y, l, r *and guard* p, *we have that if*

$$\bar{p}l = \bar{p}l\bar{p} \tag{85}$$

$$\bar{p}r \sqsubseteq r\bar{p} \tag{86}$$

$$(l \sqcap m)py \sqsubseteq py(l \sqcap m) \tag{87}$$

$$(l \sqcap m)\bar{p}y \sqsubseteq \bar{p}y(l \sqcap m) \tag{88}$$

$$(l \sqcap m)r \sqsubseteq r(l \sqcap m) \tag{89}$$

$$(py \sqcap m)r \sqsubseteq r(py \sqcap m) \tag{90}$$

$$(l \sqcap m) \quad \textit{is continuous} \tag{91}$$

$$r^\omega = r^* \tag{92}$$

then

$$\bar{p}(y \sqcap l \sqcap r)^\omega\bar{p} = \bar{p}(py \sqcap \bar{p}y \sqcap \bar{p}l \sqcap \bar{p}r\bar{p})^\omega\bar{p}.$$

Proof. Proof of refinement in one direction ($\sqsubseteq$) is trivial, so we show

$\bar{p}(y \sqcap l \sqcap r)^\omega\bar{p}$
$= \{$Proposition 7.1 using assumptions (87–89) and (91). Note that $(l \sqcap m)(y \sqcap r) \sqsubseteq (y \sqcap r)(l \sqcap m)$ follows from assumptions (87–89) and (93) below.$\}$
$\bar{p}l^\omega(y \sqcap r)^\omega\bar{p}$
$= \{$decomposition (47)$\}$
$\bar{p}l^\omega(py \sqcap r)^\omega(\bar{p}y(py \sqcap r)^\omega)^\omega\bar{p}$
$= \{$commutativity assumption (90), $r^\omega = r^*$ (92) and Proposition 7.1$\}$
$\bar{p}l^\omega(py)^\omega r^\omega(\bar{p}y(py)^\omega r^\omega)^\omega\bar{p}$
$= \{$guards are conjunctive and satisfy $g = gg$, leapfrog (48)$\}$
$\bar{p}l^\omega(py)^\omega r^\omega\bar{p}(\bar{p}y(py)^\omega r^\omega\bar{p})^\omega\bar{p}$
$\sqsupseteq \{$assumptions $r^\omega = r^*$ (92), $\bar{p}r \sqsubseteq r\bar{p}$ (86) and induction (13)$\}$
$\bar{p}l^\omega(py)^\omega\bar{p}(r\bar{p})^\omega(\bar{p}y(py)^\omega\bar{p}(r\bar{p})^\omega)^\omega\bar{p}$
$= \{$assumption $\bar{p}l = \bar{p}l\bar{p}$ (85) and (54)$\}$
$\bar{p}l^\omega\bar{p}(py)^\omega\bar{p}(r\bar{p})^\omega(\bar{p}y(py)^\omega\bar{p}(r\bar{p})^\omega)^\omega\bar{p}$
$= \{$(55), assumption $\bar{p}l = \bar{p}l\bar{p}$ (85) and (54)$\}$
$\bar{p}l^\omega(r\bar{p})^\omega(\bar{p}y(py)^\omega\bar{p}(r\bar{p})^\omega)^\omega\bar{p}$
$= \{$decomposition (47)$\}$
$\bar{p}l^\omega(\bar{p}y(py)^\omega\bar{p} \sqcap r\bar{p})^\omega\bar{p}$
$= \{$Property 7.1 using property (94) below and assumptions (87–89) and (91) on $(l \sqcap m)\}$
$\bar{p}(\bar{p}y(py)^\omega\bar{p} \sqcap l \sqcap r\bar{p})^\omega\bar{p}$
$= \{$(54) using assumptions (85) and (86), left distributivity (11), and (55) $\}$
$\bar{p}(py)^\omega((\bar{p}y \sqcap \bar{p}l \sqcap \bar{p}r\bar{p})(py)^\omega\bar{p})\bar{p}$
$= \{$guards conjunctive and leapfrog (48)$\}$
$\bar{p}(py)^\omega((\bar{p}y \sqcap \bar{p}l \sqcap \bar{p}r\bar{p})(py)^\omega)\bar{p}$
$= \{$decomposition (47)$\}$
$\bar{p}(\bar{p}y \sqcap py \sqcap \bar{p}l \sqcap \bar{p}r\bar{p})\bar{p}.$

The following results are referred to in the above proof. First we have that for any finite set of elements $\{i : 0..N \bullet y_i\}$,

$$(\forall i : 0..N \bullet xy_i \sqsubseteq y_i x) \Rightarrow x(y_0 \sqcap ... \sqcap y_N) \sqsubseteq (y_0 \sqcap .. \sqcap y_N)x. \tag{93}$$

This follows from right subdistributivity (10), the assumption $(\forall i : 0..N \bullet xy_i \sqsubseteq y_i x)$, isotony and left distributivity (11).

Using the assumptions of the theorem we also have that

$$(l \sqcap m)(y(py)^\omega \bar{p} \sqcap r\bar{p}) \sqsubseteq (y(py)^\omega \bar{p} \sqcap r\bar{p})(l \sqcap m) \tag{94}$$

may be verified by

$$\begin{array}{ll} & (l \sqcap m)(y(py)^\omega \bar{p} \sqcap r\bar{p}) \\ \sqsubseteq & \{\text{right subdistributivity (10)}\} \\ & (l \sqcap m)y(py)^\omega \bar{p} \sqcap (l \sqcap m)r\bar{p} \\ \sqsubseteq & \{\text{assumptions on } (l \sqcap m) \text{ (87–89) and (91), (56) and (57)}\} \\ & y(py)^\omega (l \sqcap m)\bar{p} \sqcap r(l \sqcap m)\bar{p} \\ \sqsubseteq & \{\text{guards conjunctive and refine 1 (31), } l \text{ does not enable } p \text{ (85), and neither does } m\} \\ & y(py)^\omega \bar{p}(l \sqcap m) \sqcap r\bar{p}(l \sqcap m) \\ = & \{\text{left distributivity (11)}\} \\ & (y(py)^\omega \bar{p} \sqcap r\bar{p})(l \sqcap m). \end{array}$$

□

The previous theorem may be generalised further to deal with a scenario where we have an extra action e, an *environment action*, which excludes the critical section of y (95), in addition to not being able to enable the critical section of y (96).

Proposition 8.5 *Using the same assumptions as Proposition 8.4, if we also have an environment action e satisfying*

$$e = \bar{p}e \quad \textit{and} \tag{95}$$

$$\bar{p}e = \bar{p}e\bar{p} \tag{96}$$

then we also have that

$$\bar{p}(e \sqcap y \sqcap l \sqcap r)^\omega \bar{p} = \bar{p}(\bar{p}e \sqcap py \sqcap \bar{p}y \sqcap \bar{p}l \sqcap \bar{p}r\bar{p})^\omega \bar{p}.$$

Proof. We have that

$$\begin{array}{ll} & \bar{p}(e \sqcap y \sqcap l \sqcap r)^\omega \bar{p} \\ = & \{\text{decomposition (47), assumptions (95,96), guards conjunctive and leapfrog (48)}\} \\ & \bar{p}(y \sqcap l \sqcap r)^\omega \bar{p}(\bar{p}e\bar{p}(y \sqcap l \sqcap r)^\omega \bar{p})^\omega \bar{p} \\ = & \{\text{Proposition 8.4 and atomicity refinement assumptions}\} \\ & \bar{p}(py \sqcap \bar{p}y \sqcap \bar{p}l \sqcap \bar{p}r\bar{p})^\omega \bar{p}(\bar{p}e\bar{p}(py \sqcap \bar{p}y \sqcap \bar{p}l \sqcap \bar{p}r\bar{p})^\omega \bar{p})^\omega \bar{p} \\ = & \{\text{assumptions (95,96), guards conjunctive and leapfrog (48)}\} \\ & \bar{p}(py \sqcap \bar{p}y \sqcap \bar{p}l \sqcap \bar{p}r\bar{p})^\omega (e(py \sqcap \bar{p}y \sqcap \bar{p}l \sqcap \bar{p}r\bar{p})^\omega)^\omega \bar{p} \\ = & \{\text{decomposition (47)}\} \\ & \bar{p}(\bar{p}e \sqcap py \sqcap \bar{p}y \sqcap \bar{p}l \sqcap \bar{p}r\bar{p})^\omega \bar{p}. \end{array}$$

□

8.3.2. *Atomicity refinement rules for action systems*

Proposition 8.5 can then be used to justify atomicity refinement rules for probabilistic action systems. For example, if we take elements ia and b such that

$$a = (\overline{\epsilon\, b})a \text{ and}$$

$$i = i(\overline{\epsilon\, b})$$

and we instantiate p and y such that

$$p = \epsilon\, b \text{ and}$$

$$y = a \sqcap b$$

then if the assumptions of Proposition 8.5 hold we can verify that

$$i \;\mathsf{do}\; e \sqcap a \sqcap b \sqcap l \sqcap r \;\mathsf{od} = i \;\mathsf{do}\; e \sqcap (a \;\mathsf{do}\; b \;\mathsf{od}) \sqcap l \sqcap r \;\mathsf{od}.$$

On the other hand if we define $y = a \sqcap b$ such that

$$a = (\overline{\epsilon\, b})a, \tag{97}$$

$$b = bp \text{ and} \tag{98}$$

$$a = a\bar{p}, \tag{99}$$

and introduce an initialisation element $i = i\bar{p}$, then

$$i \text{ do } e \sqcap a \sqcap b \sqcap l \sqcap r \text{ od} = i \text{ do } e \sqcap (\text{do } b \text{ od } a) \sqcap l \sqcap r) \text{ od}$$

follows from the assumptions of Proposition 8.5.

9. Concluding remarks and outlook

In this article we have investigated and developed the theory of a very general algebra which may be used to identify commonalities between a range of program models. Importantly, the algebra contains operators for reasoning about both finite and possibly infinite iterations, which means that it may be useful for reasoning about reactive systems, as well as non-reactive programs with a total-correctness semantics. We have particularly emphasised the role that the general refinement algebra can play in reasoning about probabilistic programs. The algebra describes core features of probabilistic programs which may be used to explain and verify a range of practical transformation theorems in a model-independent way.

This work may be seen as an extension of the earlier work of McIver et al. [MW05, MCM06] and Meinicke and Hayes [MH06, MH08b]. In their work, McIver et al. [MW05, MCM06] used a similar algebra, probabilistic Kleene algebra, for reasoning about a separation and reduction theorem for probabilistic programs. Unlike our work, theirs only considers finite iterations. Meinicke and Hayes [MH06, MH08b] used algebraic methods to reason about probabilistic program transformations using possibly infinite iterations, however they worked within a chosen probabilistic model. In this work we have lifted some of the key findings of Meinicke and Hayes to an abstract algebraic level: incorporating them into a more general theory, thereby demonstrating their validity over a range of models. We have also derived new results in the algebra. In particular we have specified and verified separation and reduction theorems which are applicable to probabilistic programs. As well as emphasising the benefits of reasoning about probabilistic programs using such a simple algebra, we have explained its limitations.

Acknowledgments

The authors are grateful to R.J.R. Back, Jules Desharnais, Ian J. Hayes, E.C.R. Hehner, Bernhard Möller and Graeme Smith for comments. This research was partially supported by the EU project DEPLOY.

Appendix A: Expectation transformers

Two expectation-transformer models for the general refinement algebra (and its specialisation scc. gRA) are given, and guards and assertions, and the enabledness and termination operators are interpreted in terms of expectation transformers. Some additional properties of these models—that were referred to earlier in the text—are also considered.

Appendix A.1. Expectation transformers and correctness reasoning

We use the one-bounded expectation-transformer theory of McIver and Morgan [MM01a, MM01b], which generalises the original expectation-transformer theory which appears in [MMS96, MM05], so that miraculous program behaviour—and hence guards—can be expressed. We first define expectations and expectation transformers, and then consider healthiness conditions.

A.1.1. Expectations

Let Σ be a state space. Then a function

$$\phi : \Sigma \to \mathbb{R}_{\geq 0}$$

is an *expectation*.

A function

$$\phi : \Sigma \to [0, 1]$$

is a *one-bounded* expectation. Given a state space Σ, we denote the set of one-bounded expectations by $\mathcal{E}_1\Sigma$. A function

$$p : \Sigma \to \{0, 1\}$$

is called a *predicate* and we denote the set of predicates over a state space Σ by $\mathcal{P}\Sigma$. We can use 1 to represent the Boolean value true and 0 to represent false. We shall use ϕ and ψ to denote expectations, ϕ_1 to denote a one-bounded expectation and p and q to denote predicates—p is, however, sometimes overloaded to denote a probability or a probability function. States will be refered to by σ.

We define eight operators on expectations, given by the following for any state $\sigma \in \Sigma$ and any probability $p \in [0, 1]$:

$$\begin{array}{ll}
(\phi \sqcap \psi).\sigma & =_{df} \mathsf{min}\{\phi.\sigma, \psi.\sigma\}, \\
(\phi \sqcup \psi).\sigma & =_{df} \mathsf{max}\{\phi.\sigma, \psi.\sigma\}, \\
(\phi + \psi).\sigma & =_{df} \phi.\sigma + \psi.\sigma, \\
(\neg\phi_1).\sigma & =_{df} 1 - \phi_1.\sigma, \\
(\phi \times \psi).\sigma & =_{df} \phi.\sigma \times \psi.\sigma, \\
(c * \phi).\sigma & =_{df} c \times \phi.\sigma, \\
(\phi \ominus c').\sigma & =_{df} \mathsf{max}\{\phi.\sigma - c', 0\} \text{ and} \\
(\phi \ {}_p\!\oplus \ \psi).\sigma & =_{df} p \times \phi + (1 - p) \times \psi,
\end{array}$$

where, when applied to reals, + denotes addition, − denotes subtraction, × denotes multiplication, and min and max denote the minimum and the maximum, respectively. We also define an order on the expectations, given by

$$\phi \leq \psi \Leftrightarrow_{df} (\forall \sigma \in \Sigma \bullet \phi.\sigma \leq \psi.\sigma).$$

The following notation for predicates,

$$\begin{array}{ll}
p \wedge q & =_{df} \ p \sqcap q, \\
p \vee q & =_{df} \ p \sqcup q, \\
\mathsf{True} & =_{df} \ (\lambda\, \sigma \in \Sigma \bullet 1), \\
\mathsf{False} & =_{df} \ (\lambda\, \sigma \in \Sigma \bullet 0) \text{ and} \\
p \Rightarrow q & \Leftrightarrow_{df} \ p \leq q,
\end{array}$$

is introduced for familiarity.

A.1.2. Expectation transformers

A *one-bounded expectation transformer* (which we often refer to simply as an expectation transformer) is a function

$$S : \mathcal{E}_1\Sigma \to \mathcal{E}_1\Sigma,$$

where Σ is any state space. One-bounded expectation transformers are the probabilistic counterpart to predicate transformers [DS90, Dij76]. In a program intuition, $S.\phi$ then denotes the *least* expectation that S will produce the expectation ϕ. More specifically, given an initial state σ, $S.\phi.\sigma$ denotes the least average value of ϕ that may be observed by executing S from σ.

There are three named expectation transformers

$$\begin{array}{rl}
\mathsf{abort} & =_{df} \ (\lambda\, \phi \bullet \mathsf{False}), \\
\mathsf{magic} & =_{df} \ (\lambda\, \phi \bullet \mathsf{True}) \text{ and} \\
\mathsf{skip} & =_{df} \ (\lambda\, \phi \bullet \phi),
\end{array}$$

and an expectation transformer S *is refined by* T, written $S \sqsubseteq T$, if

$$(\forall \phi \in \mathcal{E}\Sigma \bullet S.\phi \leq T.\phi).$$

This paper deals with six operations on expectation transformers defined by

$$\begin{aligned}
(S;\ T).\phi &=_{df} S.(T.\phi),\\
(S \sqcap T).\phi &=_{df} S.\phi \sqcap T.\phi,\\
(S \sqcup T).\phi &=_{df} S.\phi \sqcup T.\phi,\\
(S\ {}_p\!\oplus\ T).\phi &=_{df} (\lambda\, \sigma \in \Sigma \bullet S.\phi\ {}_{p.\sigma}\!\oplus\ T.\phi),\\
S^* &=_{df} \nu.(\lambda X \bullet S;\ X \sqcap \mathsf{skip}) \text{ and}\\
S^\omega &=_{df} \mu.(\lambda X \bullet S;\ X \sqcap \mathsf{skip}),
\end{aligned}$$

for any $\phi \in \mathcal{E}\Sigma$ and $p : \Sigma \to [0, 1]$, where μ and ν denote the least and the greatest fixpoint with respect to $\sqsubseteq$, respectively.

The set of one-bounded expectation transformers forms a complete lattice with least element abort, and greatest element magic.

A.1.3. Healthiness conditions

We use so called *healthiness conditions* to identify two main classes of one-bounded expectation transformers: the *deterministic* and the *dually nondeterministic* expectation transformers. These may be seen as the probabilistic counterpart to the classes of *conjunctive* and *isotone* predicate transformers, respectively. That is, the nondeterministic one-bounded expectation transformers are suitable for modelling probabilistic programs which may include discrete probabilistic choices, in addition to demonic nondeterministic choices (in fact, for finite state spaces, they characterise a relational model for such probabilistic programs, as shown in [MM01a]), whereas the set of dually nondeterministic one-bounded expectation transformers is more general, also allowing for angelic choices to be expressed. The healthiness conditions isotony, semi-sublinearity, and $\oplus$-subdistributivity are used in the definition of these terms.

One-bounded expectation transformer S is *isotone* if for all one-bounded expectations ϕ and ψ,

$$\phi \le \psi \Rightarrow S.\phi \le S.\psi.$$

It is *semi-sublinear* if

$$c * S.\phi \ominus c' \le S.(c * \phi \ominus c'),$$

for all ϕ, ψ, and non-negative constants c and c' such that $c - c' \le 1$, and it is $\oplus$-*subdistributive* if

$$S.\phi\ {}_p\!\oplus\ S.\psi \le S.(\phi\ {}_p\!\oplus\ \psi)$$

holds for all ϕ, ψ and probability $p \in [0..1]$.

A one-bounded expectation transformer that is isotone, and semi-sublinear will be called *dually nondeterministic*, and a dually nondeterministic expectation transformer which is also $\oplus$-subdistributive will be called *nondeterministic*. Also, we have that an expectation transformer is (finitely) *conjunctive* if for all one-bounded expectations ϕ and ψ,

$$S.(\phi \sqcap \psi) = S.\phi \sqcap S.\psi$$

holds, and it is *continuous* if for all directed sets of one-bounded expectations $\mathcal{B}$,

$$S.(\sqcup_{\beta:\mathcal{B}} \bullet \beta) = (\sqcup_{\beta:\mathcal{B}} \bullet S.\beta),$$

where a set $\mathcal{B}$ of expectations is *directed* if $(\forall\, \alpha, \beta : \mathcal{B} \bullet (\exists\, \gamma : \mathcal{B} \bullet \alpha \sqsubseteq \gamma \wedge \beta \sqsubseteq \gamma))$.

Appendix A.2. Soundness

Let NTran_Σ and DNTran_Σ be, respectively, the set of nondeterministic and dually nondeterministic expectation transformers over a state space Σ. From the model-based work of Meinicke and Hayes [MH08b], we readily have that both $(\mathsf{NTran}_\Sigma, \sqcap, ;\ , ^\omega, ^*, \mathsf{magic}, \mathsf{skip})$ and $(\mathsf{DNTran}_\Sigma, \sqcap, ;\ , ^\omega, ^*, \mathsf{magic}, \mathsf{skip})$ are gRA, and that these algebras are also scc. gRA for the case where Σ is finite. In this section we show how the healthiness conditions, guards and assertions, and the enabledness and termination operators from the abstract algebra can be given an expectation-transformer interpretation.

A.2.1. Healthiness conditions

The expectation-transformer healthiness condition conjunctivity, readily implies the abstract-algebraic expression of the property (27), but even more specifically, it can be shown that (27) uniquely *characterises* the set of conjunctive expectation transformers. We can also show that a *continuous*, nondeterministic expectation transformer satisfies the continuity condition (28). To verify this we require the following proposition

$$S^\omega;\ U = (\mu X \bullet S;\ X \sqcap U) \tag{100}$$

which has been verified in [MH08b]. Assume then that S, T, U and V are nondeterministic (and thus isotone) and that R is nondeterministic and continuous. Then $(\lambda X \bullet R;\ X)$ is also continuous and we can derive:

$$\begin{array}{ll} & R;\ S^\omega;\ U \sqsubseteq T^\omega;\ V;\ R \\ \Leftrightarrow & \{\text{Property (100)}\} \\ & R;\ (\mu\,.(\lambda X \bullet S;\ X \sqcap U)) \sqsubseteq \mu\,.(\lambda X \bullet T;\ X \sqcap V;\ R) \\ \Leftarrow & \{\text{fusion}\} \\ & (\lambda X \bullet R;\ X) \circ (\lambda X \bullet S;\ X \sqcap U) \sqsubseteq (\lambda X \bullet T;\ X \sqcap V;\ R) \circ (\lambda X \bullet R;\ X) \\ \Leftrightarrow & \{\text{function composition}\} \\ & (\lambda X \bullet R;\ (S;\ X \sqcap U)) \sqsubseteq (\lambda X \bullet T;\ R;\ X \sqcap V;\ R) \\ \Leftrightarrow & \{\text{refinement defined pointwise over functions}\} \\ & (\forall X \bullet R;\ (S;\ X \sqcap U) \sqsubseteq T;\ R;\ X \sqcap V;\ R). \end{array}$$

We have not yet been able to answer the question whether or not the continuity condition uniquely characterises continuity (in the sense that a nondeterministic expectation transformer is continuous if and only if it satisfies the condition).

A.2.2. Guards and assertions

Consider the function $[\cdot] : \mathcal{P}\Sigma \to (\mathcal{E}_1\Sigma \to \mathcal{E}_1\Sigma)$ such that when $p \in \mathcal{P}\Sigma$ and $\phi \in \mathcal{E}_1\Sigma$

$$[p].\phi =_{df} p \times \phi + \neg p \times \mathsf{True} = p \times \phi + \neg p.$$

These expectation transformers are called *guards*. There is also a dual, an *assertion*, and it is defined by

$$\{p\}.\phi =_{df} p \times \phi + \neg p \times \mathsf{False} = p \times \phi.$$

Complement $\bar{}$ is defined on guards and assertions by $\overline{[p]} = [\neg p]$ and $\overline{\{p\}} = \{\neg p\}$. It follows directly from the definitions that guards are conjunctive, and it is easily established that

$$(\mathsf{EGrd}_\Sigma, \sqcap, ;\ , \bar{}, \mathsf{skip}, \mathsf{magic})$$

is a Boolean algebra, where $\sqcap$ is meet, ; is join, and $\bar{}$ is complement, and EGrd_Σ is the set of (expectation transformer) guards over a state space Σ. For example, if $g \in \mathsf{EGrd}_\Sigma$, then $g \sqcap \bar{g} = \mathsf{skip}$ as the following shows: Let $[p]$ be any guard and $\phi \in \mathcal{E}_1\Sigma$. Then

$$\begin{array}{ll} & ([p] \sqcap [\neg p]).\phi \\ = & \{\text{definition of } \sqcap\} \\ & [p].\phi \sqcap [\neg p].\phi \\ = & \{\text{definition of guards, double negation}\} \\ & (p \times \phi + \neg p) \sqcap (\neg p \times \phi + p) \\ = & \{p \text{ predicate}\} \\ & \phi \\ = & \{\text{definition of } \mathsf{skip}\} \\ & \mathsf{skip}.\phi. \end{array}$$

The rest of the axioms for Boolean algebra are verified similarly and guards are easily seen to be closed under $\bar{}$,$\sqcap$ and ;. Moreover, guards are a subset of the nondeterministic one-bounded expectation transformers (and hence a subset of the dually nondeterministic transformers).

It is also easy to argue that the assertions in the predicate-transformer sense are a model for assertions in the abstract-algebraic sense.

A.2.3. Enabledness and termination

Enabledness. The *miracle guard*, gd, is defined as a mapping from a one-bounded expectation transformer to a predicate by

$$\mathsf{gd}.S =_{df} (\lambda\sigma \in \Sigma \bullet S.\mathsf{False}.\sigma \neq 1)$$

in [MH08b]. Intuitively, $\mathsf{gd}.S$ is a predicate that specifies those start states from which S does not certainly behave miraculously. For any dually nondeterministic S, we interpret ϵS as $[\mathsf{gd}.S]$. This definition satisfies the probabilistic enabledness axioms (33) and (34). For the case where the underlying state space is finite, it additionally satisfies enabledness axiom (35).

The first enabledness axiom ($\epsilon xx = x$) has been verified in [MH08b]. To see that the second ($g \sqsubseteq \epsilon(gx)$) is valid, first note that

$$\begin{array}{ll}
& [p] \sqsubseteq [\mathsf{gd}.([p];\ S)] \\
\Leftrightarrow & \{\text{basic property of guards: } [p] \sqsubseteq [q] \text{ iff } q \Rightarrow p\} \\
& \mathsf{gd}.([p];\ S) \Rightarrow p \\
\Leftrightarrow & \{\text{definition of gd}\} \\
& (\lambda\,\sigma \bullet ([p];\ S).\mathsf{False}.\sigma \neq 1) \Rightarrow p \\
\Leftrightarrow & \{\text{definition of } [\cdot] \text{ and } ;\} \\
& (\lambda\,\sigma \bullet p.\sigma \times S.\mathsf{False}.\sigma + \neg p.\sigma \neq 1) \Rightarrow p \\
\Leftrightarrow & \{\text{functions defined pointwise}\} \\
& (\lambda\,\sigma \bullet (p.\sigma \times S.\mathsf{False}.\sigma + \neg p.\sigma \neq 1) \Rightarrow p.\sigma)
\end{array}$$

and then consider the two cases $p.\sigma = 1$ and $p.\sigma = 0$. In the case $p.\sigma = 1$ the implication trivially holds, since true is implied by anything. In the case $p.\sigma = 0$ the left-hand side of the implication becomes false ($1 \neq 1$), and so the implication holds by the fact that false implies anything.

The validity of the third axiom ($\epsilon(xy) = \epsilon(x\epsilon y)$), which is dependent on the finite state space assumption, needs a more elaborate argument. First note that

$$\begin{array}{ll}
& [\mathsf{gd}.(S;\ T)] = [\mathsf{gd}.(S;\ [\mathsf{gd}.T])] \\
\Leftrightarrow & \{\text{basic guard properties}\} \\
& \mathsf{gd}.(S;\ T) = \mathsf{gd}.(S;\ [\mathsf{gd}.T]) \\
\Leftrightarrow & \{\text{definition of gd}\} \\
& (\lambda\,\sigma \bullet (S;\ T).\mathsf{False}.\sigma \neq 1) = (\lambda\,\sigma \bullet (S;\ [\mathsf{gd}.T]).\mathsf{False}.\sigma \neq 1) \\
\Leftrightarrow & \{\text{definition of sequential composition and guard}\} \\
& (\lambda\,\sigma \bullet S.(T.\mathsf{False}).\sigma \neq 1) = (\lambda\,\sigma \bullet S.(\mathsf{gd}.T \times \mathsf{False} + \neg\mathsf{gd}.T).\sigma \neq 1) \\
\Leftrightarrow & \{\text{definition of gd}\} \\
& (\lambda\,\sigma \bullet S.(\mathsf{gd}.T \times T.\mathsf{False} + \neg\mathsf{gd}.T).\sigma \neq 1) = \\
& (\lambda\,\sigma \bullet S.(\mathsf{gd}.T \times \mathsf{False} + \neg\mathsf{gd}.T).\sigma \neq 1).
\end{array}$$

The equation immediately above may then be verified as follows. Let

$$\begin{array}{l}
\phi = (\mathsf{gd}.T \times T.\mathsf{False} + \neg\mathsf{gd}.T), \\
\psi = (\mathsf{gd}.T \times \mathsf{False} + \neg\mathsf{gd}.T) \text{ and} \\
\beta = (x * \mathsf{gd}.T + \neg\mathsf{gd}.T),
\end{array}$$

where $x =_{df} (\sqcup\sigma \in \mathsf{gd}.T \bullet T.\mathsf{False}.\sigma)$. Since Σ is finite, from the definition of gd we have that $0 \leq x < 1$. We also have that $\psi \leq \phi \leq \beta$.

Since $\psi \leq \phi$, from isotony, and the fact that expectations are bounded above by 1 we have that $(\forall\,\sigma \bullet (S.\phi.\sigma \neq 1) \Rightarrow (S.\psi.\sigma \neq 1))$. Verifying implication in the other direction is more complex. By definition $\phi \leq \beta$, so, from isotony and bounds on expectations, we have that it is sufficient to show that

$$(\forall\,\sigma \bullet (S.\beta.\sigma \neq 1) \Leftarrow (S.\psi.\sigma \neq 1)).$$

Let $c = \frac{1}{1-x}$, $c' = \frac{x}{1-x}$. Since $0 \leq x < 1$, these constants are well defined and satisfy $c \geq 0$, $c' \geq 0$, and $c - c' = 1 \leq 1$. We then have that $c * \beta \ominus c' = \psi$, hence for any σ

$$
\begin{array}{ll}
& S.\psi.\sigma \neq 1 \\
\Leftrightarrow & \{\text{definition of constants } c \text{ and } c'\} \\
& S.(c * \beta \ominus c').\sigma \neq 1 \\
\Rightarrow & \{\text{semi-sublinearity, isotony, bounds on expectations}\} \\
& c * S.\beta.\sigma \ominus c' \neq 1.
\end{array}
$$

In order to show that this implies that $S.\beta.\sigma \neq 1$, we can perform a proof by contradiction. Assume that $c * S.\beta.\sigma \ominus c \neq 1$, and $S.\beta.\sigma = 1$. We would then have that

$$
\begin{array}{ll}
& c * S.\beta.\sigma \ominus c' \neq 1 \\
\Leftrightarrow & \{\text{assumption } S.\beta.\sigma = 1\} \\
& c * 1 \ominus c' \neq 1 \\
\Leftrightarrow & \{\text{definition of } c \text{ and } c'\} \\
& 1 \neq 1 \\
\Leftrightarrow & \{\text{logic}\} \\
& \mathsf{false}.
\end{array}
$$

For infinite state spaces, it is possible to show that, for the set of dually nondeterministic one-bounded expectation transformers, the third enabledness axiom (35) does *not* hold:

Example A.1 Take an infinite state space Σ in which a variable x is defined and has type $\mathbb{N}$, and let S and T be of type DNTran_Σ where

$$
\begin{array}{l}
S =_{df} \mu\,.(\lambda X \bullet x := x + 1;\ X \sqcup 1) \text{ and} \\
T =_{df} (\mathsf{abort}\ {}_{(\frac{1}{x+1})}\oplus\ \mathsf{magic}).
\end{array}
$$

If we define σ_0 to be of type Σ such that $\sigma_0.x = 0$, then

$$
\begin{array}{ll}
& (S;\ T).\mathsf{False}.\sigma_0 \\
= & \{\text{definition of } T, \mathsf{magic} \text{ and } \mathsf{abort}\} \\
& S.(1 - \frac{1}{x+1}).\sigma_0 \\
= & \{\text{definition of } S \text{ and } \sigma_0\} \\
& (\sqcup i \in \mathbb{N}_{>0} \bullet 1 - \frac{1}{i}) \\
= & 1.
\end{array}
$$

However, since $(\forall \sigma \in \Sigma \bullet T.\mathsf{False}.\sigma \neq 1)$,

$$
\begin{array}{ll}
& (S;\ [\mathsf{gd}.T]).\mathsf{False}.\sigma_0 \\
= & \{\text{definition of } T \text{ and } \mathsf{gd}\} \\
& S;\ [\mathsf{True}].\mathsf{False}.\sigma_0 \\
= & \{\text{definition of } \mathsf{skip}\} \\
& S.\mathsf{False}.\sigma_0 \\
= & \{\text{definition of } S\} \\
& (\sqcup i \in \mathbb{N} \bullet (x := x + i).\mathsf{False}.\sigma_0) \\
= & (\sqcup i \in \mathbb{N} \bullet 0) \\
= & 0,
\end{array}
$$

and so $[\mathsf{gd}.(S;\ T)]$ is not equal to $[\mathsf{gd}.(S;\ [\mathsf{gd}.T])]$.

It is unknown if a counter example exists for the smaller class of nondeterministic expectation transformers when the state space is infinite.

The following counterexample demonstrates that property $\epsilon x \bot = x \bot$ (enabledness axiom (36) in the non-probabilistic case), does not necessarily hold.

Example A.2 Consider the program $(x := 1)_{\frac{1}{2}} \oplus\ \mathsf{magic}$, which is always enabled. Then

$$
((x := 1)_{\frac{1}{2}} \oplus\ \mathsf{magic});\ \mathsf{abort} = (\mathsf{abort}\ {}_{\frac{1}{2}}\oplus\ \mathsf{magic}),
$$

whereas

$$[\mathsf{gd}.((x := 1) \mathbin{{}_{\frac{1}{2}}\!\oplus} \mathsf{magic})];\ \mathsf{abort} = [\mathsf{True}];\ \mathsf{abort} = \mathsf{abort},$$

and programs $(\mathsf{abort} \mathbin{{}_{\frac{1}{2}}\!\oplus} \mathsf{magic})$ and abort are not equal.

Termination. The *termination guard* of a one-bounded expectation transformer S, $\mathsf{term}.S$, is defined as

$$(\lambda\sigma \in \Sigma \bullet S.\mathsf{True}.\sigma \neq 0).$$

The termination guard of an expectation transformer S is a standard predicate which specifies those states from which S will not certainly abort. For any dually nondeterministic transformer S, we interpret τS as $\{\mathsf{term}.S\}$. This definition satisfies the termination axioms (39), (40), and (42). For example, the following proof demonstrates that (40) is sound:

$$\begin{array}{ll}
& \{\mathsf{term}.(\{p\};\ S)\} \sqsubseteq \{p\} \\
\Leftrightarrow & \{\text{basic assertion properties}\} \\
& \mathsf{term}.(\{p\};\ S) \Rightarrow p \\
\Leftrightarrow & \{\text{definition of termination guard}\} \\
& (\lambda\,\sigma \bullet (\{p\};\ S).\mathsf{True}.\sigma \neq 0) \Rightarrow p \\
\Leftrightarrow & \{\text{definition of sequential composition, assertion}\} \\
& (\lambda\,\sigma \bullet p.\sigma \times (S.\mathsf{True}.\sigma \neq 0) + \neg p.\sigma \times (0 \neq 0)) \Rightarrow p \\
\Leftrightarrow & \{\text{ordering on expectations defined pointwise}\} \\
& (\forall\,\sigma \bullet p.\sigma \times (S.\mathsf{True}.\sigma \neq 0) \Rightarrow p.\sigma) \\
\Leftrightarrow & \{\text{logic}\} \\
& \mathsf{true}.
\end{array}$$

For the case where the state space is finite, termination axiom (41) is also satisfied. (This may be verified in a similar fashion to the corresponding enabledness axiom (35).) The following example (cf. Example A.1) demonstrates the dependence of termination axiom (41) on the finite state space assumption.

Example A.3 Let Σ be an infinite state space containing variable x of type $\mathbb{N}$, and $S,\ T \in \mathsf{NTran}_\Sigma$, where

$$\begin{array}{l}
S =_{df} (x := x + 1)^* \text{ and} \\
T =_{df} (\mathsf{magic} \mathbin{{}_{\frac{1}{x+1}}\!\oplus} \mathsf{abort}).
\end{array}$$

We have that $\{\mathsf{term}.(S;\ T)\}$ is not equal to $\{\mathsf{term}.(S;\ \{\mathsf{term}.T\})\}$.

A.3. Total and weak correctness assertions

In this section we verify that the property (45)

$$(\mathsf{magic} = [p];\ S;\ [\neg q]) \Rightarrow ([p];\ S;\ [q] = [p];\ S).$$

holds for any nondeterministic one-bounded expectation transformer S, and predicates p and q, and we explain why it does not necessarily hold if S is dually nondeterministic.

First we observe that, for the nondeterministic expectation transformers, the algebraic total and weak correctness properties $\mathsf{magic} = [p];\ S;\ [\neg q]$ and $[p];\ S;\ [q] = [p];\ S$ may be related to expectation transformers in the following way.

Proposition A.4 *For nondeterministic expectation transformer S and predicates p and q,*

$$[p];\ S;\ [q] = [p];\ S \Leftarrow p \leq S.q \quad \textit{and} \tag{101}$$

$$[p];\ S;\ [\neg q] = \mathsf{magic} \Leftrightarrow p \leq S.q. \tag{102}$$

Proof. This proof uses the property (recall the equivalences from Sect. 6)

$$[p];\ S;\ [q] = [p];\ S \Leftrightarrow \{p\};\ S;\ \{q\} = \{p\};\ S. \tag{103}$$

Assume that S is a nondeterministic expectation transformer. First, to prove (101) it can be seen that

$$
\begin{array}{ll}
& [p]; S; [q] = [p]; S \\
\Leftrightarrow & \{(103), \text{refinement ordering}\} \\
& (\forall \phi \bullet (\{p\};\ S;\ \{q\}).\phi = (\{p\};\ S).\phi) \\
\Leftrightarrow & \{\text{definition of sequential composition, assertion}\} \\
& (\forall \phi \bullet p \times S.(q \times \phi) = p \times S.\phi) \\
\Leftrightarrow & \{\text{isotony}\} \\
& (\forall \phi \bullet p \times S.(q \times \phi) \geq p \times S.\phi).
\end{array}
$$

If we assume that $p \leq S.q$ then this property may be verified as follows. For any expectation ϕ and state σ such that $p.\sigma$

$$
\begin{array}{ll}
& S.(q \times \phi).\sigma \\
= & \{\text{expectations are one-bounded}\} \\
& S.(q + q \times \phi + (\neg q) \times \phi \ominus 1).\sigma \\
= & S.(1 * q + 1 * \phi \ominus 1).\sigma \\
\geq & \{\text{from the assumption that } s \text{ is an nondeterministic one-bounded} \\
& \text{expectation transformer we have that for all non-negative constants} \\
& c_1, c_2, c, \text{ if } c_1 + c_2 - c \leq 1 \text{ and one-bounded expectations } \phi \text{ and } \psi, \\
& c_1 * S.\phi + c_2 * S.\psi \ominus c \leq S.(c_1 * \phi + c_2 * \psi \ominus c)\ \} \\
& S.q.\sigma + S.\phi.\sigma \ominus 1 \\
= & \{S.q.\sigma = 1 \text{ follows from assumption } p \leq S.q\} \\
& S.\phi.\sigma.
\end{array}
$$

To prove property (102) we show the more general result that for any expectation transformer S and expectations ϕ and ψ, we have that

$$[\neg\psi];\ \mathsf{abort} \sqsubseteq S;\ [\neg\phi] \Leftrightarrow \psi \leq S.\phi, \tag{104}$$

where $[\neg\psi]$ and $[\neg\phi]$ are *probabilistic guards* $\mathsf{magic}\ {}_{\psi}\oplus\ \mathsf{skip}$ and $\mathsf{magic}\ {}_{\phi}\oplus\ \mathsf{skip}$, respectively. This is shown simply by

$$
\begin{array}{ll}
& [\neg\psi];\ \mathsf{abort} \sqsubseteq S;\ [\neg\phi] \\
\Leftrightarrow & \{\text{isotony of sequential composition and transitivity of } \sqsubseteq\} \\
& [\neg\psi];\ \mathsf{abort} \sqsubseteq S;\ [\neg\phi];\ \mathsf{abort} \\
\Leftrightarrow & \{\text{refinement ordering definition}\} \\
& (\forall \theta \bullet ([\neg\psi];\ \mathsf{abort}).\theta \leq (S;\ [\neg\phi];\ \mathsf{abort}).\theta) \\
\Leftrightarrow & \{\text{definition of sequential composition and abort}\} \\
& [\neg\psi].\mathsf{False} \leq S.([\neg\phi].\mathsf{False}) \\
\Leftrightarrow & \{\text{definition of probabilistic guard}\} \\
& \psi \leq S.\phi.
\end{array}
$$

Since, for expectation transformer S and predicates p and q, $([p];\ S;\ [\neg q] = \mathsf{magic})$ and $([\neg p];\ \mathsf{abort} \sqsubseteq S;\ [\neg q])$ are equivalent (see Sect. 6), this proves our result. □

Property (45)—which is also satisfied by the conjunctive predicate transformers [vW02]—is an immediate consequence of this result.

Corollary A.5 For nondeterministic expectation transformer S and predicates p and q

$$[p];\ S;\ [\neg q] = \mathsf{magic} \Rightarrow [p];\ S;\ [q] = [p];\ S$$

holds. □

As for the isotone predicate transformers [vW04], this property does not hold for the dually nondeterministic expectation transformers: when S is dually nondeterministic, property (102) from Theorem A.4 continues to hold but (101) is—in general—lost. Consider $S =_{df} x := 1 \sqcup x := 2$, $p = \mathsf{True}$, $q = (\lambda\sigma \bullet \sigma.x = 1)$. The reason that this relationship is lost is that assertions of the form $[p];\ S;\ [q] = [p];\ S$ fail to accurately represent weak-correctness assertions (involving predicates) in the angelic models.

References

[Bac89] Back RJR (1989) A method for refining atomicity in parallel algorithms. In: PARLE'89 conference on parallel architectures and languages Europe. Lecture notes in computer science, vol 366, pp 199–216. Springer, Berlin

[BKS83] Back RJR, Kurki-Suonio R (1983) Decentralization of process nets with centralized control. In: Proceedings of the 2nd ACM SIGACT-SIGOPS symposium on principles of distributed computing. ACM Press, New York, pp 131–142

[BvW98] Back RJR, von Wright J (1998) Refinement calculus: a systematic introduction. Springer, Berlin

[BvW99] Back RJR, von Wright J (1999) Reasoning algebraically about loops. Acta Inform 36(4):295–334

[Coh00] Cohen E (2000) Separation and reduction. In: Mathematics of program construction. Lecture notes in computer science, vol 1837, pp 45–59. Springer, Berlin

[Dij76] Dijkstra EW (1976) A discipline of programming. Prentice-Hall, Englewood Cliffs

[DMS06] Desharnais J, Möller B, Struth G (2006) Kleene algebra with domain. ACM Trans Comput Logic 7(4):798–833

[Doe77] Doeppner TW (1977) Parallel program correctness through refinement. In: ACM symposium on principles of programming languages. ACM, New York, pp 155–169

[DP90] Davey BA, Priestley HA (1990) Introduction to lattices and order. Cambridge University Press, Cambridge

[DS90] Dijkstra EW, Scholten CS (1990) Predicate calculus and program semantics. Springer, Berlin

[Ehm03] Ehm T (2003) The Kleene algebra of nested pointer structures: theory and applications. PhD thesis, University of Augsburg, December 2003

[HS08] Höfner P, Struth G (2008) Can refinement be automated? Electron Notes Theor Comput Sci 201:197–222

[Koz90] Kozen D (1990) On Kleene algebras and closed semirings. In: MFCS '90: proceedings on mathematical foundations of computer science. Lecture notes in computer science, vol 452, pp 26–47. Springer, Berlin

[Koz94] Kozen D (1994) A completeness theorem for Kleene algebras and the algebra of regular events. Inform Comput 110(2):366–390

[Koz97] Kozen D (1997) Kleene algebra with tests. ACM Trans Program Lang Syst 19(3):427–443

[Lip75] Lipton RJ (1975) Reduction: a method of proving properties of parallel programs. Commun ACM 18(12):717–721

[LS89] Lamport L, Schneider FB (1989) Pretending atomicity. Technical report, Ithaca, NY, USA

[MÖ4] Möller B (2004) Lazy Kleene algebra. In: Mathematics of program construction. Lecture notes in computer science, vol 3125, pp 252–273. Springer, Berlin

[MCM06] McIver AK, Cohen E, Morgan CC (2006) Using probabilistic Kleene algebra for protocol verification. In: Relations and Kleene algebra in computer science. Lecture notes in computer science, vol 4136, pp 296–310

[Mei08] Meinicke LA (2008) Transformation rules for probabilistic programs: an algebraic approach. PhD thesis, The University of Queensland, June 2008

[MH06] Meinicke LA, Hayes IJ (2006) Reasoning algebraically about probabilistic loops. In: Eighth internatinal conference on formal engineering methods. Lecture notes in computer science, vol 4260.

[MH08a] Meinicke L, Hayes IJ (2008) Probabilistic choice in refinement algebra. In Audebaud P, Paulin-Mohring C (eds) Mathematics of program construction. Lecture notes in computer science, vol 5133, pp 243–267. Springer, Berlin

[MH08b] Meinicke LA, Hayes IJ (2008) Algebraic reasoning for probabilistic action systems and while-loops. Acta Inform 45(5):321–382

[MM01a] Morgan C, McIver A (2001) Cost analysis of games, using program logic. In: APSEC '01: proceedings of the eighth Asia-Pacific on software engineering conference, p 351. IEEE Computer Society, Washington

[MM01b] Morgan C, McIver A (2001) Cost analysis of games using program logic. http://www.cse.unsw.edu.au/~carrollm/probs/bibliography.html

[MM05] McIver A, Morgan C (2005) Abstraction, refinement and proof for probabilistic systems. Monographs in Computer Science. Springer, Berlin

[MMS96] Morgan C, McIver A, Seidel K (1996) Probabilistic predicate transformers. ACM Trans Program Lang Syst 18(3):325–353

[MS08] Meinicke L, Solin K (2008) Reactive probabilistic programs and refinement algebra. In: Berghammer R, Möller B, Struth G (eds) Relations and Kleene algebra in computer science. Lecture notes in computer science, vol 4988, pp 304–319. Springer, Berlin

[MW05] McIver A, Weber T (2005) Towards automated proof support for probabilistic distributed systems. In: Sutcliffe G, Voronkov A (eds) Logic for programming, artificial intelligence, and reasoning, 12th international conference. Lecture notes in computer science, vol 3835, pp 534–548. Springer, Berlin

[PK00] Patron MC, Kozen D (2000) Certification of compiler optimizations using Kleene algebra with tests. In: Lloyd J, Dahl V, Furbach U, Kerber M, Lau K, Palamidessi C, Moniz Pereira L, Sagiv Y, Stuckey PJ (eds) Proceedings of the 1st international conference in computational logic. Lecture notes in artificial intelligence, vol 186, pp 568–582. Springer, Berlin

[Sol06] Solin K (2006) On two dually nondeterministic refinement algebras. In: Relations and Kleene algebra in computer science. Lecture notes in computer science, vol 4136, pp 373–387

[ST96] Sere K, Troubitsyna E (1996) Probabilities in action systems. In: Proceedings of the 8th Nordic workshop on programming theory. Publishing Association, Helsinki

[SvW06] Solin K, von Wright J (2006) Refinement algebra with operators for enabledness and termination. In: Mathematics of program construction. Lecture notes in computer science, vol 4014, pp 397–415. Springer, Berlin (revised version to appear in Science of Computer Programming)

[TF06] Takai T, Furusawa H (2006) Monodic tree Kleene algebra. In: Relations and Kleene algebra in computer science. Lecture notes in computer science, vol 4136, pp 402–416

[vW02] von Wright J (2002) From Kleene algebra to refinement algebra. In: Mathematics of program construction. Lecture notes in computer science, vol 2386, pp 233–262. Springer, Berlin

[vW04] von Wright J (2004) Towards a refinement algebra. Sci Comput Program 51:23–45

Received 9 January 2008

Accepted in revised form 20 December 2008 by E.A. Boiten, M.J. Butler, J. Derrick and G. Smith

Published online 17 April 2009